

AD-A132 211

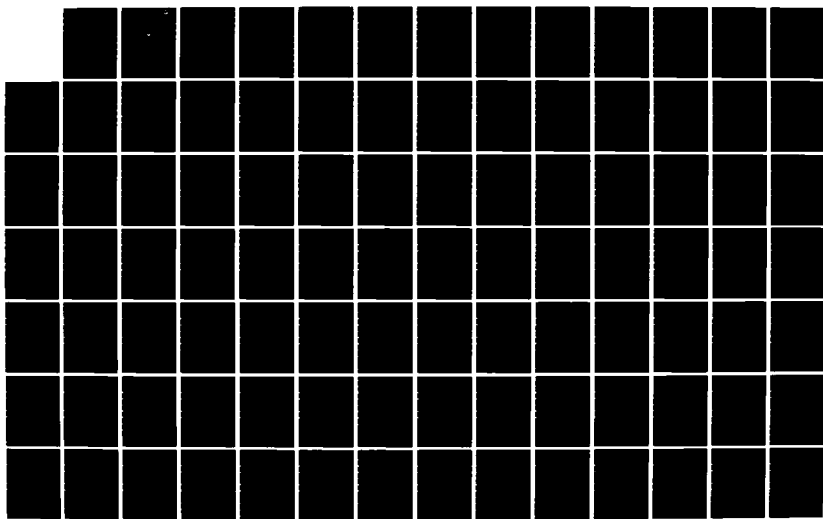
KNOWLEDGE BASE MANAGEMENT FOR MODEL MANAGEMENT SYSTEMS
(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA G W WATSON
JUN 83

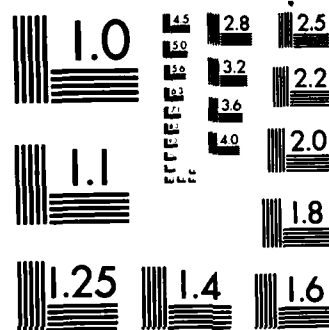
1/2

UNCLASSIFIED

F/G 5/1

NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

ADA 132211

2

NAVAL POSTGRADUATE SCHOOL

Monterey, California



SEP 3 1983
S
D

THESIS

KNOWLEDGE BASE MANAGEMENT
FOR MODEL MANAGEMENT SYSTEMS

by

George William Watson, Jr.

June 1983

Thesis Advisor:

Daniel D. Dolk

Approved for public release, distribution unlimited.

83 09 28 019

DTIC FILE COPY

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. A132211	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Knowledge Base Management for Model Management Systems		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis June 1983
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) George William Watson, Jr.		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		12. REPORT DATE June 1983
		13. NUMBER OF PAGES 110
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release, distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Decision Support Systems, Knowledge Management, Model Management		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) This study examines the issues involved in bringing qualitative and quantitative techniques to bear upon unstructured managerial decisions. Furthermore, this work reviews the problems of user interface and data base interfaces as they relate to aspects of model base managements. The focus of this study is to identify some organizations of knowledge about models within the Decision Support System.		

DD FORM 1473
1 JAN 73

EDITION OF 1 NOV 68 IS OBSOLETE
S/N 0102-LF-014-6601

Unclassified
1 SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

In support of this goal, this report investigates what knowledge is, how it is structured, and how it is accessed.

Accession For	
NTIS GRA&I	<input checked="checked" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution	
A	



Unclassified

Approved for public release, distribution unlimited.

Knowledge Base Management for Model Management Systems

by

George William Watson, Jr.
Lieutenant, United States Marine Corps
B.A., University of Washington, 1978
M.B.A., California State University, Fullerton, CA, 1981

Submitted in partial fulfillment of the
requirements for the degree of

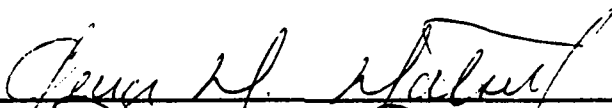
MASTER OF SCIENCE IN INFORMATION SYSTEMS


from the


NAVAL POSTGRADUATE SCHOOL
June 1983


Author:

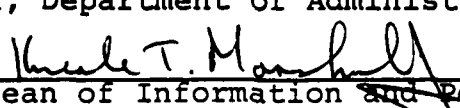
Approved by:





Thesis Advisor


Second Reader


Chairman, Department of Administrative Sciences


Dean of Information and Policy Sciences

ABSTRACT

This study examines the issues involved in bringing qualitative and quantitative techniques to bear upon unstructured managerial decisions. Furthermore, this work reviews the problems of user interface and data base interfaces as they relate to aspects of model base management.

The focus of this study is to identify some organizations of knowledge about models within the Decision Support System. In support of this goal, this report investigates what knowledge is, how it is structured, and how it is accessed.

TABLE OF CONTENTS

I.	INTRODUCTION -----	9
II.	THE NATURE OF THE DECISION PROCESS -----	12
	A. COGNITIVE STYLES -----	12
	B. ORGANIZATIONAL FRAMEWORKS FOR ANALYSIS -----	14
	C. TYPES AND STRUCTURES OF DECISIONS -----	17
	D. HUMAN INFORMATION PROCESSING -----	20
	E. SUMMARY -----	21
III.	DEFINITION OF THE DECISION SUPPORT SYSTEM -----	24
	A. TYPICAL DECISION SUPPORT SYSTEM PARADIGMS ---	24
	B. KNOWLEDGED-BASED SYSTEMS -----	32
	C. ELEMENTS OF MODEL BASE MANAGEMENT -----	34
IV.	A SURVEY OF ARTIFICIAL INTELLIGENCE	
	FOR MODEL BASE MANAGEMENT SYSTEMS -----	39
	A. REPRESENTING KNOWLEDGE	
	USING PRODUCTION RULES -----	39
	B. SEMANTIC NETWORKS -----	42
	C. REPRESENTING A MODEL BASE WITH FRAMES -----	49
	D. FRAMES AND ABSTRACTIONS -----	54
V.	ORGANIZATION OF KNOWLEDGE ABOUT THE MODEL BASE --	59
	A. LEVELS OF ABSTACTION -----	59
	B. COMPARING FRAMES AND ABSTRACTIONS -----	63
	C. COMBINING FRAMES AND ABSTRACTIONS -----	65
	D. THE MODEL DOMAIN -----	69
	E. FRAME SYSTEM FUNCTIONS -----	75
	F. ENGLISH-LIKE COMMAND SYSTEM -----	80

VI. CONCLUSIONS -----	87
A. AREAS FOR FURTHER RESEARCH -----	87
B. IMPACT ON THE DSS PARADIGM -----	89
C. SUMMARY -----	91
APPENDIX A: PROGRAM LISTINGS FOR THE FRAME SYSTEM ---	93
APPENDIX B: PROGRAM LISTINGS FOR ENGLISH-LIKE COMMANDS -----	99
APPENDIX C: USER'S GUIDE -----	107
LIST OF REFERENCES -----	108
INITIAL DISTRIBUTION LIST -----	110

LIST OF FIGURES

2.1	Important Elements in the Decision Process	-----	22
3.1	Sprague and Carlson's Paradigm for a DDS	-----	25
3.2	Sprague and Carlson's Implementation Technology	--	28
3.3	A DSS with Knowledge Base Support	-----	31
4.1	A Fragment of a Semantic Network		
	Representing a Model Base	-----	44
4.2	A Sample Network of User Requirements	-----	45
4.3	A Sample Network Seeking		
	the Transportation Model	-----	46
4.4	Building Property Lists		
	and Retrieving Their Value	-----	48
4.5	A Model Base of Frames	-----	51
4.6	The General Structure of a Frame	-----	53
4.7	The General Structure of an Abstraction	-----	56
5.1	Logical and Occurrence Structure of a Model Base	--	61
5.2	Comparing Frames and Abstractions	-----	66
5.3	Convention for Describing a Frame-like		
	Abstraction	-----	70
5.4	Organization of the Frame System	-----	72
5.5	Example of a Lower Level Frame	-----	73
5.6	Example of a Higher Level Frame	-----	74
5.7	Example of an Input Script	-----	81
5.8	Schematic of the Augmented Transition Network	---	83
5.9	The Four Phases of Answering the English Query	--	84

5.10	General Structure of a Parse-tree	-----	86
6.1	A DSS Paradigm with Knowledge Base Support	-----	90

I. INTRODUCTION

This study examines the topic of Model Base Management in Decision Support Systems (DSS). Model Management concepts involve bringing appropriate quantitative and qualitative techniques to bear upon structured, semistructured, and unstructured managerial decisions. During the decision-making session, the effective DSS should assist the user (unfamiliar with computer science or management science) in identifying, sequencing, and executing those models appropriate to the problem instance. A myriad of issues arise concerning the human interface with this type of system. These problems include speaking to the user in terms not excessively technical, and linking the organization's data base with the model base. The fields of management science, Artificial Intelligence (AI), computational linguistics, and psychology are involved in providing theoretical resolutions to some of these problems.

In particular, this work describes the issue of a knowledge base in support of the DSS. Its intent is to identify some appropriate organizations on knowledge about models. In support of this objective it is necessary to examine general issues about human decisionmaking in organizational contexts. In addition, an investigation is made about characteristics of the information in the knowledge base: how much there is, how it is structured, and how

it can be accessed. Finally, this study will discuss areas and implications for further research.

Several conceptual points provided the foundation upon which this work is built. First, Decision Support Systems are differentiated from typical management information system applications by the nature of their task. The DSS focuses on unstructured problems whose specifications are unclear or incomplete. Upper level strategic planners and managers daily face problems with these characteristics. DSS primarily are targeted to support these planners and managers.

The traditional aspects of data retrieval and access remain vital considerations in DSS. In the DSS, much of the data manipulation and analysis is accomplished by another segment of the software system called the "model base". This aspect relieves the user from post-session, manual data manipulation. This feature makes the DSS much more flexible, adaptable, and friendly than the typical Management Information System (MIS) or Data Based Management System (DBMS) applications.

A DSS is flexible when it is able to address a wide range of unstructured and underspecified problems. It is adaptable when it can approach the problem resolution from many directions to accomodate varying problemsolving styles. It is, therefore, the entire objective of the DSS to improve the performance and productivity of people that rely on information to make decisions.

The work presented here is structured as follows.

Chapter II is a survey of the classic and recent literature addressing the salient issues of the DSS field. Chapter III attempts to outline the prominent views of exactly what constitutes a DSS. Chapter IV surveys the representational techniques from the field of AI which shows promise for the DSS field. Chapter V is a description of an actual implementation of a knowledge base about models, and some English-like commands that facilitate its use. Chapter VI summarizes the results and conclusions that can be drawn from this work.

II. THE NATURE OF THE DECISION PROCESS

The study of computer-aided decisionmaking, and in particular DSS, requires a review of theory regarding the nature of human decision processes. This review is important for two reasons: (1) it aids in designing a DSS compatible with human beings, and (2), it establishes a frame of reference for the presentation of this study. This section discusses the decision process and the decision-maker, including cognitive styles, organizational frameworks, procedures and theories regarding human information processing, and decision structures.

A. COGNITIVE STYLES

The manner in which humans perceive and arrive at a frame of reference, or a problem solution, has been the subject of much psychological research. It has been claimed that an inhibiting factor in the application of quantitative methods to problems is the fact that managers and management scientists think differently [Ref. 1].

McKenney and Keen [Ref. 2] present four styles of human cognition. Cognitive style refers to the method by which humans organize the information they perceive. This style is, in part, habitual, but is developed through training and experience. The authors define these styles as perceptive, receptive, intuitive, and systemic. Preceptive thinkers

are those that focus on relationships between data. Preceptive people look for how things differ and conform. These similarities and differences cue conclusions about which data to gather, and which decisions to make. Receptive thinkers focus on details of a situation rather than overviews. Receptive thinkers take a bottom-up approach to constructing the problem instance. The perceptive thinker will take an overview, or top-down, approach to problem formulation. Therefore, the receptive thinker's conclusions are drawn from analysis rather than precepts. Systemic thinkers tend to structure their problem instance to a solution algorithm; that is, they look at the problem in terms of how it fits into an existing model solution. On the other hand, intuitive thinkers are sensitive to clues that may not be verbalized or overt. Intuitive thinkers include and discard information in a trial-and-error fashion.

The above models for cognitive styles are somewhat supported by other researchers. Edward deBono [Ref. 3] discusses the concept of vertical and lateral thinking. Vertical thinking is loosely related to systemic thinking, where one idea establishes a logical foundation upon which to construct the next idea. Solution of a calculus problem exemplifies this style, as one rule is applied to reach a state upon which another rule is applied, until the solution is reached. Lateral thinking attempts to escape from this regimen. Lateral thinkers seek out information from

perceived circumstances. Lateral thinking is somewhat associated with creative thinking, and the idea of pattern matching from one circumstance to another.

Mintzberg [Ref. 4] has related these concepts to the critical human activities of planning and managing. He asserts the process of managing is a logical and analytical endeavor which seeks out the structure and reason in a scenario. However, he believes management is a highly intuitive and creative process. The manager, for example, is more sensitive to unspoken signals, gestures, and underlying trends in the scenario. Due to the dynamics and randomness of organizational settings, "hunches" and "holistic perceptions" play important roles. Managers must deal with such intangibles as morale, whereas planners deal with forecasted numbers.

It is apparent that any automated decision aid is likely to have both systemic and intuitive users. A system which pathologically attempts to structure the way a decision is made is likely to invoke frustration and contempt, thereby defeating whatever good characteristics are endemic to the DSS.

B. ORGANIZATIONAL FRAMEWORKS FOR ANALYSIS

With this characterization of the nature of individual thought as a background, this section will attempt to place the individual inside the organizational setting. At this

juncture we are broadening the scope of the DSS, reaching beyond individual cognition to organizational functioning. There is, in effect, an interactive and dynamic association between human decisionmakers and the organizational structure.

Researchers have identified several dimensions to the phenomenon of organizational decisionmaking. Roland [Ref. 5] speaks of the group, the environment, the task, the situation, the individual, and the available technology as having impacts on how information is gathered and used.

Huber [Ref. 6] discusses four organizational decision models and calls them the decision environment. These are the rational model, the political model, the garbage can model, and the program model. Huber defines the rational model as an environment where organizational units use information in an intentionally rational manner to make decisions for the organization. The political model depicts an environment where organizational decisions are consequences of the application of strategy and tactics by persons or units seeking to influence decision processes in favor of themselves.

The garbage can model is somewhat more abstract. In this theory, a "can" is a choice situation, or an opportunity to resolve a problem. To have a can, people must perceive a problem. The garbage can model discusses a decision as the consequence of problems looking for solutions, solutions looking for problems, and opportunities

for making decisions. The program model sees two reasons for organizational decisions being what they are: (1) decisions are constrained by standard operating procedures; and (2), programming, training, and experience reinforce past decision processes which, in turn, affect future decisions (history constrains choice).

Huber goes on to say that certain types of information are required for each model. The rational model, for example, looks at several types of data in great detail in an effort to project the logical decision. The political model, however, looks less at data, and more at the people involved in the situation, and how they can be influenced. The important question in the garbage can model is "what are the problems and opportunities that are present in the organization"?

The program model infers that the organization can be understood best by looking at several key elements of information. The first element is historical trend; trends from the past can be projected into the future. Another element of importance is the experience of the organization; experience creates a pattern of behavior and policy. Standard operating procedures also are important elements in discovering the organizational priorities and concerns.

The reader may notice a certain parallel between organizational models and cognitive styles. A systemic and receptive person may find it easier if the work environment

follows the rational model. On the other hand, intuitive and perceptive thinkers may experience greater success in a garbage can business.

Most organizations exhibit characteristics of all of these models. A comprehensive DSS will accommodate the breadth of models and cognitive styles presented here.

C. TYPES AND STRUCTURES OF DECISIONS

DSS typically are touted as supporting unstructured decisionmaking. The literature, however, is somewhat vague in defining exactly the differences between structured and unstructured decisions, and which problem instances are structured and which are not. Some authors contend that if an algorithm can be placed upon the parameters, then it is a structured decision. Others contend that unstructured decisions can be transformed into structured ones [Ref. 7].

Stabell [Ref. 8] discusses three dimensions to the structure of a decision. The first dimension relates structure to alternatives. The degree of structure is determined by the ease with which alternative solutions can be identified, i.e., the easier it is to generate solutions, the greater the structure. Consider, for example, an office building with hundreds of employees, and a very overworked elevator system providing access to its 50 floors of office space. Employees are late for work, late going to lunch, and late returning. This adversely affects

morale and productivity. Given that the structure of the building will not allow for the addition of any more elevators, and the lift capacity already is optimized, what are the tenants to do? This is an example of a problem where structured models have contributed all they can to optimize lift capacity. It is quite difficult to generate solution paths through this problem, and is, therefore, unstructured according to the parameter of alternatives.

Let us look at this problem from another of Stabell's dimensions, task predictability. In an unstructured task it is difficult to predict the consequences of the decision to be made. It may be difficult to trace through the chain of events that are apt to occur. The more unstructured the task, the greater the number of variables relevant to the task solution. In our example, we might consider staggering work hours and lunch hours, or perhaps add escalators in the stairwells (if possible), but can we predict with any degree of certainty the relative impact of these decisions?

This leads to Stabell's final dimension--that of epistemic uncertainty: that is, which variables are important and relevant to the task solution? This is, perhaps, the most difficult question to answer. Returning to our example, which is in fact an actual case, consider the subsequent solution which illustrates the difficulty in predicting important variables. The overriding difficulty with the elevator overload was the impatience of

those waiting to get to his or her destination. When the owner put mirrors on the walls around the elevators, complaints dropped off drastically. Evidently people now were occupied by watching themselves and others while waiting for the elevators.

Structured decisions give clear clues as to which variables and solution algorithms are needed. These might include certain short term forecasting functions, and linear optimization problems which are deterministic in nature. Unstructured tasks include mergers, portfolio management, and new product development--each of which have complex variable interactions. Of course, semistructured tasks share aspects from both ends of the spectrum. Production scheduling, for example, involves some structured aspects for optimizing production, and some unstructured aspects such as inventory ramifications and input availability.

The preceding discussion is designed to illustrate the dimensions of structure to a decision or problem instance. Gorry and Scott Morton [Ref. 9] point out the functional contexts, or types of decisions, which were implied earlier: operational control, management control, and strategic planning decisions. Each of these can vary with regard to structure, organizational framework, and individual style.

Independent of cognitive style is the theory of human information processing. The next section provides a short

discussion on the classical theory of how humans retrieve and store information they process.

D. HUMAN INFORMATION PROCESSING

Newell and Simon [Ref. 10] provide the classical theory of human information processing. Most theories of human information processing include several steps: gathering information, problem identification, generating alternatives, prioritizing solutions, evaluating alternatives, implementing actions, and evaluating feedback. This is the well-known intelligence, design and choice, which Newell and Simon describe as a serial process which makes use of small short term memories, and large long term memories--both of which have varying "read" and "write" times. As a result, the authors contend that the problem solving process is somewhat related to a heuristic search algorithm in AI. Specifically, the problem solver has some goal in mind, and a current condition. By comparing them, one can generate steps that help move from the current state toward the goal. Take, for example, Rubik's Cube. A certain number of moves lead toward the ultimate solution, and the problem solver assesses the current condition of the cube, with the goal being solid colors on all six sides. This is sometimes called means-ends analysis.

An important contribution of this theory is the characterization of the memories involved (the processor is discussed under cognitive styles). Short term memory

can be written to quickly, but cannot retain much information (five to seven elements). A phone number is a good example. After looking it up, one must repeat it several times on the way to the phone. Writing to long term memory requires a longer memorization process, but once accomplished, it stays in memory a long while.

E. SUMMARY

Four dimensions of the decision process have been discussed: cognitive style, organizational frameworks, the structure of decisions, and the manner in which humans process information (Figure 2.1). Each of these dimensions in the decision process are important when considering design of the DSS. A system which has strengths in one area may not be used due to weaknesses in another. Take, for example, MYCIN [Ref. 11]: most experts agree that this system diagnoses infections very well, but it is used very little. This is due, in part, to the importance of making a correct diagnosis when choosing a course of action. Can it be that the organizational framework is inappropriate for such a system? Perhaps expert systems are best suited for places, and organizations where experts are not available.

Figure 2.1 summarizes some of the elements important in the design and implementation of DSS. These elements are cornerstones of success for the DSS. Although somewhat intangible and nonquantifiable, these elements must be

DESIGN FEATURE	KEY CONSIDERATION
COGNITIVE STYLE	PRECEPTIVE, RECEPTIVE SYSTEMIC, INTUITIVE
ORGANIZATIONAL FRAMEWORK	RATIONAL, POLITICAL, GARBAGE CAN
STRUCTURE	ALTERNATIVES, EPISTEMIC UNCERTAIN PREDICTABILITY
TYPES OF DECISIONS	STRUCTURED, SEMI- STRUCTURED, UNSTRUC
HUMAN INFORMATION PROCESSING	INTELLIGENCE, DESIGN CHOICE, MEMORIES

Figure 2.1: Important Elements in the Decision Process.

considered in the physical implementation described in the next section.

III. DEFINITION OF THE DECISION SUPPORT SYSTEM

The term Decision Support System (DSS) has been used to refer to a wide range of decision assisting tools. The list may include a manager flipping a coin; an executive work station where one can speak in a natural language to a battery of display devices; or a system with the ability to dynamically formulate, resolve, and present several illustrations of a problem solution (numerical versus graphical output). The only fundamental consensus that runs through the field of DSS is that they should assist humans in making decisions.

A. TYPICAL DECISION SUPPORT SYSTEM PARADIGMS

There are two important paradigms within the body of literature on DSS. Each has an important framework, yet each illustrates a distinct difference. Sprague and Carlson [Ref. 12] present a very comprehensive paradigm which is presented in Figure 3.1. These authors define the DSS as an interactive computer-based system that helps decision-makers utilize data and models to solve unstructured problems. They see a DSS as three subsystems:

- (1) A Dialog Management System (DGMS),
- (2) A Model Base Management System (MBMS), and,
- (3) A Data Base Management System (DBMS).

They further divide these subsystems into various components. The Dialog Management System, for example,

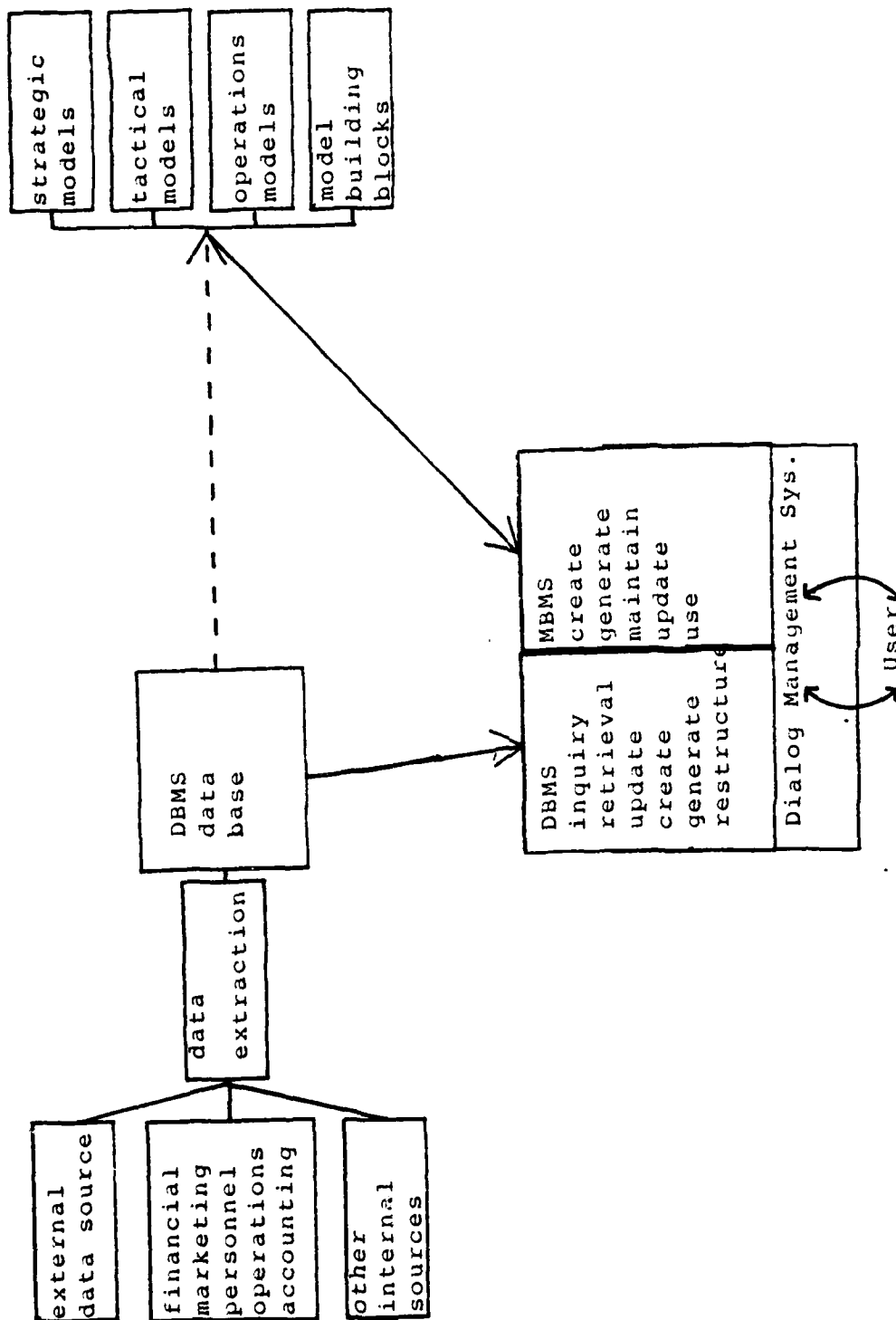


Figure 3.1: Sprague and Carlson's Paradigm for a DSS. [Ref. 12]

includes a display or presentation language, such as the graphics that the user sees. The DGMS also includes the input language or facilities (trackball, mouse, light-pen). An important feature of the DGMS is the knowledge base, or what the user needs to know to communicate with the system.

The Model Base Management System provides model management capabilities by integrating a wide range of quantitative techniques for all levels of management. Further, the MBMS must provide for:

- (1) The creation of new models;
- (2) the access to existing models;
- (3) the integration of existing models;
- (4) the modification of specific model blocks;
- (5) the modification of multi-module models;
- (6) the cataloging of the model base; and,
- (7) the linking of the models to the data base.

The DSS demands full DBMS capabilities. Data extraction must be rapid and flexible enough to respond to unanticipated user requests. Other important features include the ability to elicit data from a variety of sources, and the ability to add or remove data sources. The DBMS also should handle unofficial data from experimental sources, as well as having the ability to portray data structures in understandable form to the user, such as a relational data structure.

In the above discussion there is no mention of how the system is invoked to solve the user's specific problem. It is in the implementation that conceptual differences are most readily observed. Sprague and Carlson rely on an evolutionary development of a DSS. Each increment is designed and built to satisfy a specific user problem instance, or group of instances (i.e., a financial analysis package). In practice, the manager communicates his specifications to an implementor, who determines with the aid of a toolsmith and a builder which models and other packages the specific DSS calls for.

Implicit in this scheme of implementation are the following assumptions:

- (1) the manager is aware of the exact problem;
- (2) the intermediaries (builders and toolsmiths) are aware of the solution techniques for the problem; and
- (3) the problem characteristics are static enough to be predicted prior to design and development.

Problems with these characteristics are likely to be somewhat structured. This may contradict the definition that the DSS is for unstructured decisionmaking (for Sprague and Carlson's implementation procedure). Figure 3.2 illustrates the authors' implementation method.

A second paradigm for DSS is discussed in Bonczek, et. al [Ref. 13], who view the DSS as consisting of three subsystems. These elements are a Language Subsystem (LS), Knowledge Subsystem (KS), and a Problem Processing Subsystem (PPS).

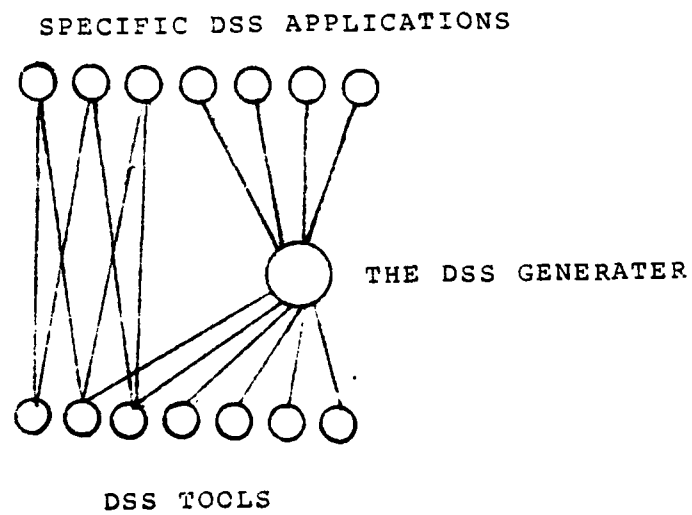


Figure 3.2: Sprague and Carlson's
Implementation Technology.
[Ref. 12]

The LS is meant to encompass the total of all linguistic facilities (retrieval and computational languages). This perspective appears to eliminate vehicles other than a language from the LS. An important aspect of the LS is that it can be designed so that the user is unaware of whether he is directing a data retrieval or model construction process.

The authors remark that unless the DSS contains knowledge about the decisionmaker's problem domain the system will be of little practical value. Therefore, the KS contains facts that the user need not retain. This concept echoes Sprague and Carlson's knowledge base, but differs significantly in prominence within the paradigm. It prescribes that the knowledge base be an important part of the DSS.

The PPS is a mechanism which relates input from the language system to facts and heuristics about the problem domain. It is through the PPS that problems are recognized, models formulated, and data retrieved. The authors relate this aspect of the automated system to human decision-making. In essence, their perception of DSS emulates the human process of problem solving, that is, the problem must be verbalized in a manner understandable to the problem solver whose special knowledge is brought to bear on the problem solution.

This concept varies considerably in implementation from the Sprague and Carlson model. It does not assume that the

decisionmaker has extensive knowledge about the problem and future environments, and it does not insert human communication between the user and the builder or generator. As a result, it appears to be a more flexible paradigm.

This study includes paradigms to define and construct the DSS presented as in Figure 3.3. The DSS can be characterized as having multiple and easy-to-use input languages, a knowledge base that responds to these inputs to ask the user for further clarifications to identify problems, a readily accessible data base, a model base, and a problem processor that takes information from the knowledge base to construct models and to link with data to provide a system response.

In implementation, the knowledge base will hold information regarding problem instances and model applications to these instances (the problem this study addresses), so that there is no requirement that a decisionmaker foresee circumstances. This helps to make a DSS a more generalized tool.

Although it is not the intent of this study to address matters of implementation, they are important in helping to clarify the nature of their usage and, therefore, their definition. Sprague and Carlson's evolutionary approach appears to be to develop one specific DSS at a time, but not with a preconceived solution template in mind, as the knowledge base rests with the user. The foundation upon

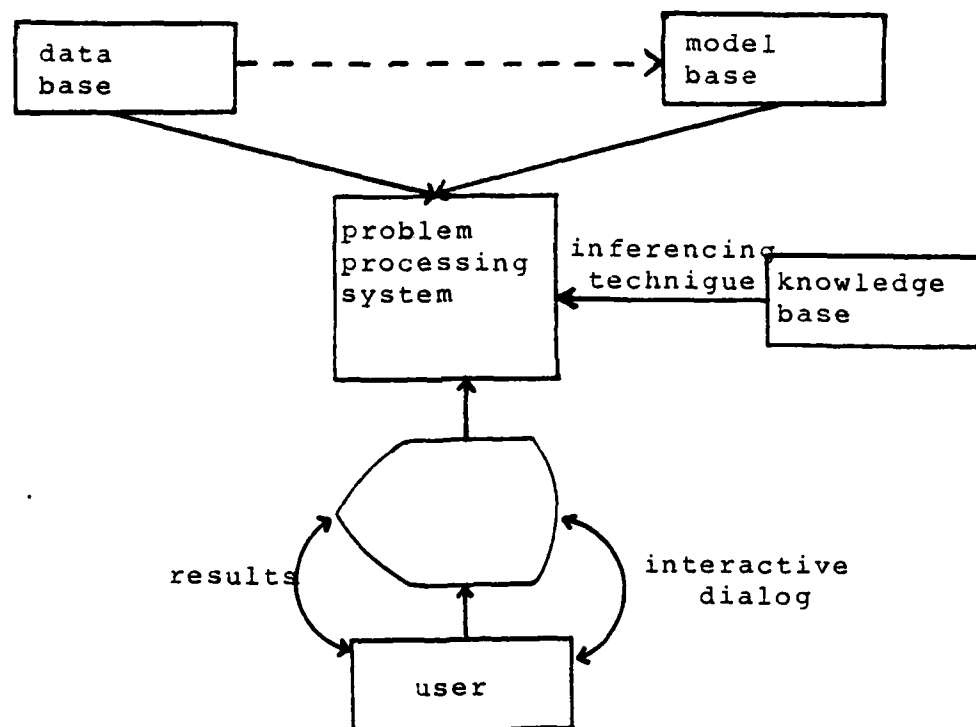


Figure 3.3: A DSS with Knowledge Base Support.

which this thesis is built calls for an evolutionary approach that facilitates a variety of solutions. The tradeoff between a specific problem and a general (generic) solution algorithm is thus established. This introduces the importance of Artificial Intelligence to the world of DSS.

B. KNOWLEDGED-BASED SYSTEMS

This section deals with the concept of knowledge-based systems. These systems are introduced here to distinguish them from DSS, as well as to lay the foundation for the following section on model base management.

In Artificial Intelligence, the terms of "knowledge-based systems" and "expert systems" are analogous. They are terms that define a computerized expert in a specific application. Such applications are widely varied, ranging from medical diagnosis to teaching aids. What each system has in common is a pool of expert knowledge to which it refers to make sound decisions.

Winston [Ref. 14] describes two important points about expert systems. The first point deals with the construction of the expert system. This includes the idea of a state, or some assessment, of the current circumstances. There also exist control variables, or transformation operators, which take as input the current state, apply some operation, and provide as output some resulting state. The concepts of iteration and recursion play important roles, as do division

of problems into subproblems, and repeated application of the transformation operators, known as recursion.

The second idea of importance contributed to DSS from expert problem solving is the question of which knowledge is important to the system. Winston calls this topic knowledge engineering, which includes the following questions: What kind of knowledge is important? Is this knowledge specialized? What are the most important ideas? How should this knowledge be represented? How much knowledge should be represented? Does a handful of facts cover 99 percent of the likely circumstances? Does comprehensive knowledge require extraordinary programming effort? And finally, exactly what information is required? For example, if the system is designed to perform calculus functions, then rules for integration and differentiation are required. In a natural language understanding system, knowledge about semantics as well as syntax is important. In other words, having the appropriate structure is not enough--the meaning of the elements play a vital role.

As Winston states:

Like compiled computer programs, memorized formulas give little opportunity for assessing the sources (of the knowledge) with the natural result being poor response to even slightly unexpected changes. . . . formulas are essences irreversibly distilled, general techniques for summary and speed.

A DSS can be seen as an expert problem solver dealing with problems that arise in a strategic environment. However, a DSS must be a more comprehensive knowledge-based

system to respond to the unexpected changes to which Winston refers. Expert problem solvers typically deal with one specific domain. MYCIN [Ref. 11], for example, deals solely with diagnosis of infections. A DSS, on the other hand, must be prepared to span several problem domains (e.g., financial, marketing, production, and personnel).

Some aspects of DSS are especially compatible with the techniques of expert systems, particularly model base management. AI techniques add an extra dimension of flexibility to the DSS. Through innovative procedures they can at least attempt to match situations and circumstances to existing data structures and processes. This is reminiscent of the chess playing algorithm. On occasion, when the human has stumped the program, an interesting phenomenon occurs--the system persists in making the best possible move. This is the type of response that must occur at all levels of the DSS. This is an effort to maximize the application of the DSS. AI techniques of pattern matching, heuristic search, and natural language understanding--all contribute significantly to the design of DSS. The following section looks in detail at model base management, an area of DSS which may be significantly enhanced by AI techniques.

C. ELEMENTS OF MODEL BASE MANAGEMENT

The focus of this study is on elements of the DSS that involve model base management. There appears to be some

consensus as to what MBMS objectives are; however, techniques for accomplishing these functions differ.

Bonczek, et. al. [Ref. 15] lists five points for the design of model support in DSS:

- (1) models should support a variety of functional areas;
- (2) models should be able to stand alone, or in a job stream with other models;
- (3) model bases should have the ability to extract data from the DBMS;
- (4) model bases should have a command language for easy interface; and,
- (5) model bases should evolve in knowledge and use ("learn").

This suggests a definition of a model base for the purpose of this study. A model base is a collection of models, where models are analogous to data in the data base. The model base will consist of models that either are user built, built by a model builder internal to the organization, or externally purchased. In addition, the model base must support a variety of tasks and analytic approaches to problem resolution. It is important to point out that some models will be building blocks in other, more comprehensive models.

Some of the objectives of the data base management system carry over to the model base management system as well:

- (1) application independence;
- (2) model update, creation, and deletion;

- (3) maintenance of a model dictionary; and,
- (4) provision of a convenient user interface.

The issues of model base management involve practical considerations as well as design theories. For example, it is one matter to advocate convenient user interface, and quite another to decide what "convenient" means.

Again, the field of AI makes important contributions to the attainment of these objectives. This contribution involves considerations about the type and organization of knowledge important to the model base. Data base administration also provides important insights about the problem of storing and accessing information about models. Combining the contributions from each field may be fruitful, especially when considering Dolk's model abstraction [Ref. 16], and Minsky's frames [Ref. 17].

Furthermore, we can discuss the specialized knowledge about each model or model component within the framework presented by Winston [Ref. 17] for expert systems. For example, the model base builder and administrator must decide how much information about each model or component is appropriate. Too much information may render the system unnecessarily cumbersome in terms of user response and overhead; on the other hand, too little information may similarly render the DSS inadequate.

In addition, the designer and the builder must determine what information is needed for model base management.

This includes issues like: how detailed and specific the knowledge should be, what generalities about classes of models should be included, and which ideas and aspects about a model instance are most important?

Besides asking what level of detail to include, one must determine which items of knowledge should be included. Should knowledge about the mathematical algorithm be included? Is it likely this knowledge will be beneficial in light of what we have said about human information processing? What about lists of applications to which the model has been, or could be, applied? Should the system be self-teaching? For example, once a new application has been identified by a user, should the list of applications be automatically updated? Finally, what types of representational techniques should be used?

In the preceding paragraphs there have been discussions and questions relating to human cognition, organizational frameworks, DSS definition, human information processing, definition of the DSS, knowledge-based systems, and model base management. The purpose of this survey is to identify the important considerations and objectives of a DSS. From this point, the scope is narrowed to model base management. In the following sections, the questions and considerations previously identified are applied to the major problem of how to represent models and knowledge about models. In essence, model management must meet many criteria (as

must DBMS and DGMS). The way models are represented has an important impact on the usefulness of the DSS.

IV. A SURVEY OF ARTIFICIAL INTELLIGENCE FOR MODEL BASE MANAGEMENT SYSTEMS

A. REPRESENTING KNOWLEDGE USING PRODUCTION RULES

As a means of representing knowledge, production systems rely on condition-action pairs called production rules. Barr and Feigenbaum [Ref. 18] assert that a production system consists of three parts:

- (1) a rule base, composed of a set of production rules;
- (2) a context, in the form of a short term memory buffer;
- (3) an interpreter which controls the system's actions.

A production rule consists of a condition part and an action part. For example, a typical production rule has the format:

IF (condition) THEN (action)

The context is essentially the list of all conditions which must be met to execute (or "fire") the action. This context changes dynamically as some actions might require the system to satisfy other conditions, all of which will reside on the context list.

The interpreter is an application specific program which decides what to do, given that certain conditions are met. For example, some tasks of the interpreter consist of: adding elements to the context list, check for duplicates on the context list, update the context list, and execute the production rules.

Some of the advantages of production rules include:

(1) Modularity--Each rule can be considered a piece of knowledge in that it states a situation and shows what should occur when that situation arises. Rules are easily added and removed. This is possible because one production rule does not call another; they merge only via the context list.

(2) Uniformity--Because production rules are all interpreted by the same interpreter, they must be in the same format. This has the advantage of being more easily understood by a person not involved in the initial development, therefore, it is somewhat self-documenting. Semantic nets, discussed in the following section, are somewhat free-form in comparison, making them more difficult to understand, and more difficult to use.

(3) Naturalness--"If-then" types of conclusions are frequently used by human experts in explaining what deductions they make in reaching conclusions. Statements saying "what to do given a particular situation" are quite natural to the human user. Semantic nets are not as precise in their relations to a situation making them slightly more cumbersome to understand by humans in a deductive situation.

Disadvantages include:

(1) Inefficiency--The advantages of modularity and uniformity require large amounts of rules to describe complex and dynamic situations. This phenomenon, known as

combinatorial explosion, creates extensive overhead in terms of run-time and memory resources. Because of the precise nature of production rules, plausible problem situations have to be preprogrammed, making them somewhat inflexible. In other words, production systems need a high degree of knowledge management and structure.

(2) Opacity--It is difficult to follow the flow of control of an interpreter firing off production rules. A high-level language, by comparison, can be traced with relative ease as it calls for subroutines.

As an example, consider a model base consisting of forecasting, optimization, probabilistic, and simulation models where optimization models consist of both nonlinear and linear models, and in turn, linear models are tied to applications such as transportation, transshipment, assignment or goal programming. The sample production rules might look like:

Production rule 1: IF (objective function is linear)
AND (linear constraints)
THEN (put linear programming model
on context list).

Production rule 2: IF (units of commodity are to be shipped)
THEN (put transportation model on context list).

The interpreter in this example separates the rules and restructures them into a new statement which, when executed,

accomplishes the action required. For example, this might involve evoking a new program composed of action parts that invoke the required model input procedures for the user.

In summary, production rule systems are useful when:

- (1) the knowledge we are trying to represent is essentially deterministic rather than heuristic;
- (2) the processes carried out can be seen as a set of independent actions; and,
- (3) each action element is relatively independent of other actions.

In terms of model base management, two factors determine whether production rules are appropriate: what knowledge will the system assume that the user has, and what knowledge will be an integral part of the system? In the above example, a user may not realize whether his constraints are linear. It may demand a lot of a production system to determine whether the problem calls for a specific model. For example, in a production system each "if-case" must be anticipated. In an unstructured setting, this is not practical. All circumstances cannot be foreseen, therefore, all rules will not make it into the data base; for this reason, production systems are more suitable for structured situations.

B. SEMANTIC NETWORKS

As a representational scheme, semantic networks have been used in many applications [Ref. 19]. Their chief

advantage is the ability to represent relationships between states or objects. Barr and Feigenbaum [Ref. 18] define the notation of semantic networks to be a series of nodes connected by arcs. Nodes normally represent objects or concepts, and arcs represent relationships between them.

Making inferences from the semantic network (or reasoning with the network as a knowledge base) has taken a variety of paths. One method is called network matching. Say, for example, we have decided to represent our model base using semantic networks (Figure 4.1). Suppose further that the user needs to know if there are any models in the model base that use such elements as "commodities shipped", "destinations", and "sources". He or she is not aware, nor particularly concerned whether the model being sought is linear. In effect, the user will develop a network fragment such as that in Figure 4.2. This fragment represents a problem instance. This instance is compared against the network representation of the model base, and matches are presented to the user. Of course, if there are no matches, an appropriate response to the user might include closest possible models.

The power in matching is the inferencing it can support when properly implemented. For example, suppose we want to ask: "are there any models that analyze transportation problems?" The network is depicted in Figure 4.3. This fragment does not exactly match the network in Figure 4.1.

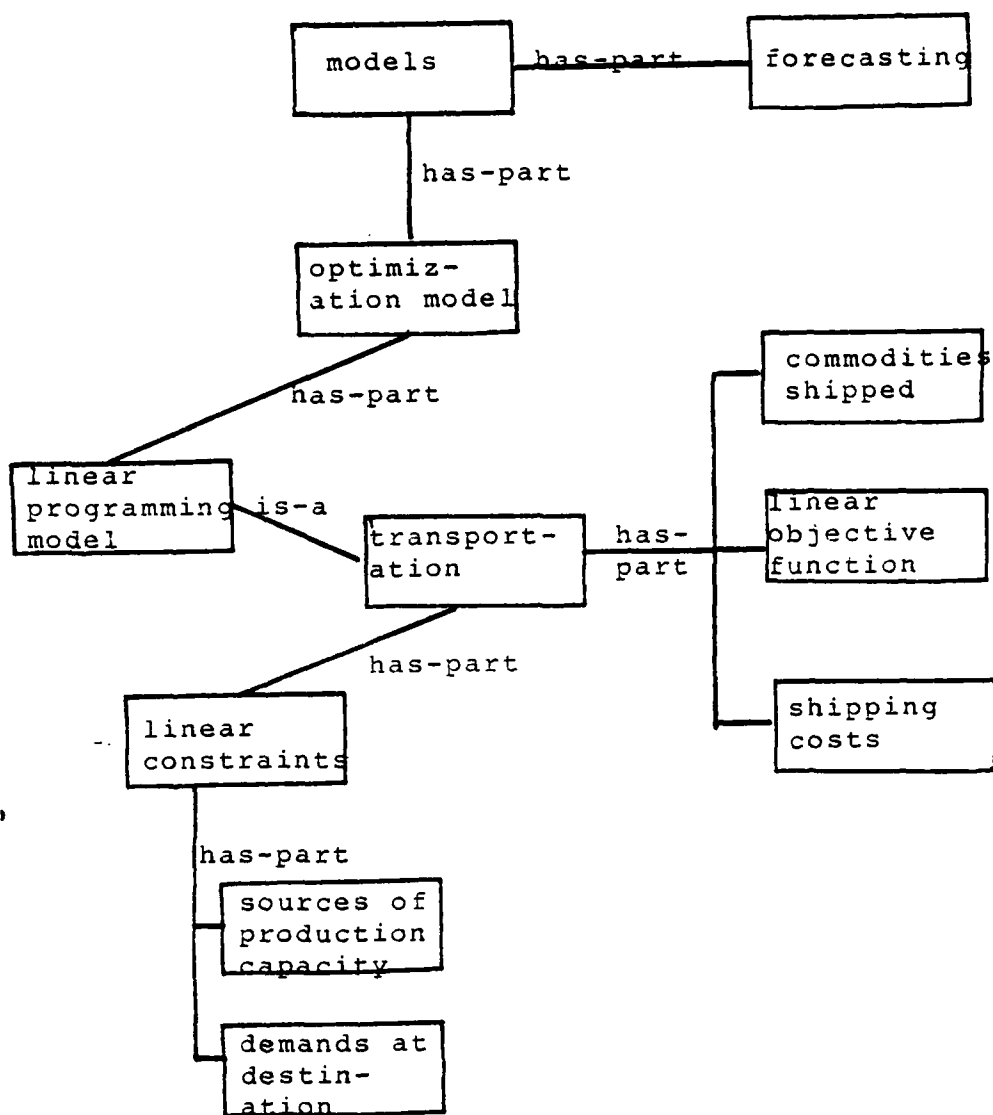


Figure 4.1: A Fragment of a Semantic Network Representing a Model Base.

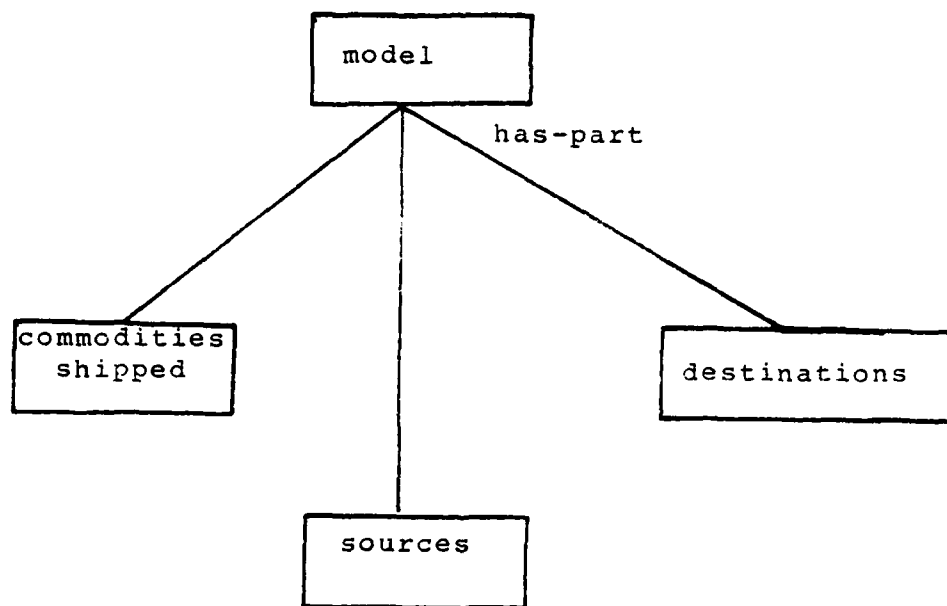


Figure 4.2: A Sample Network of User Requirements.

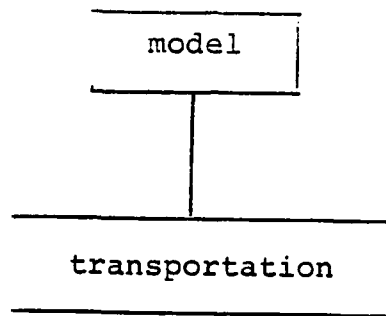


Figure 4.3: A Sample Network Seeking the Transportation Model.

The system would have to deduce, through several links, that such a model exists. Through the model's "has-part" link, optimization's "has-part" link, and linear-programming model's "is-a" link, the identification of the appropriate model is made. This is easy to accomplish from the semantic network by examining the relationships to the model node. The use of heuristics can augment this selection process.

If a language such as LISP [Ref. 20] were used to implement this knowledge base, its vehicle would be property lists. A property list contains certain information about a specific concept or object (node). On the property list of models, for example, one would attach linear programming, nonlinear programming, etc. On the list of linear programming models, one would put the values of transportation, assignment, etc. In this manner a series of is-a, has-part, and similar-to relationships are associated with certain properties. For example, the first statement in Figure 4.4 would assign to the property "model" the values "optimization", "simulation", and "forecasting". Statements two and three assign further values to the subcomponents of "model". Statements four, five, and six illustrate how to retrieve this information.

Thus, a series of LISP programs and property lists can be constructed to analyze a semantic network. A central problem in designing semantic networks, and one to be addressed in this presentation, is what properties and relationships to include. Is-a and part-of are important, but what others are important for model base management?

-
1. (putprop 'model' (optimization simulation forecasting probability) 'instance)
 2. (putprop 'optimization '(linear non-linear...) 'instance)
 3. (putprop 'linear '(transportation assignment...) 'instance)
 4. (get 'model 'instance)
 5. (get 'linear 'instance)
 6. (get 'optimization 'instance)
 7. execute the model
-

Figure 4.4: Building Property Lists and Retrieving Their Values.

Aggregate network structures are sometimes referred to as frames. The idea here is that frames will contain all of the has-part and is-a links of major conceptual parts of the network. The following discussion of frames illustrates this point.

C. REPRESENTING A MODEL BASE WITH FRAMES

Conceptually, a frame is analogous to a frame of reference. As humans we are confronted with many situations which are quite similar to previous experiences. We come to expect, for example, certain things in a place we call a restaurant. While we learn to swim, we begin to recognize the relations between movement, bouyancy, breathing, and choking on water. The next time we swim, hopefully we improve, for we have a knowledge frame of reference about our environment which becomes more detailed with experience. If a machine could capture this human quality of applying past experience to current situations, the potential for learning and problem solution might be significantly enhanced. This is the objective of Minsky's [Ref. 17] theory of frames. A frame, as defined by Minsky, is a data structure for representing stereotyped information about a situation.

Frames easily represent hierarchical information. At top levels, certain generalities are included to represent things always encountered in a particular situation--at a birthday party there is always a cake. At lower levels, sometimes called terminals, there are specific instances

that distinguish it from similar situations. A piñata, for example, might give indications of the cultural heritage of the group which could cue further searches.

Figure 4.5 illustrates the nature of the hierarchy of frames as applied to our sample model base. An "abstraction" is a notion related to a frame. It is a structure describing certain features about an object (in our case, a model). An abstraction contains such things as data objects, and assertions describe the relationships between them. Note that some information in the Figure can be related to the notion of a model abstraction. For example, knowledge about the behavior between the operators and the objects is described in the transportation model with assertions such as $C(ij)$ --being the cost of shipment of a commodity between source i and destination j .

Frame implementations are another form of property lists. The vocabulary of frames is somewhat different, however. The property list is assigned to describe a particular frame name. Properties now become identifying characteristics called slots. Slots are the general characteristics of an object or idea. Slots may take a variety of values. (Tables in a restaurant may be either French Provincial or Early American.) A slot in Figure 4.5, for example, would be one of the linear programming models.

There may be several types of algorithms to solve the linear programming models. Hillier and Lieberman [Ref. 24]

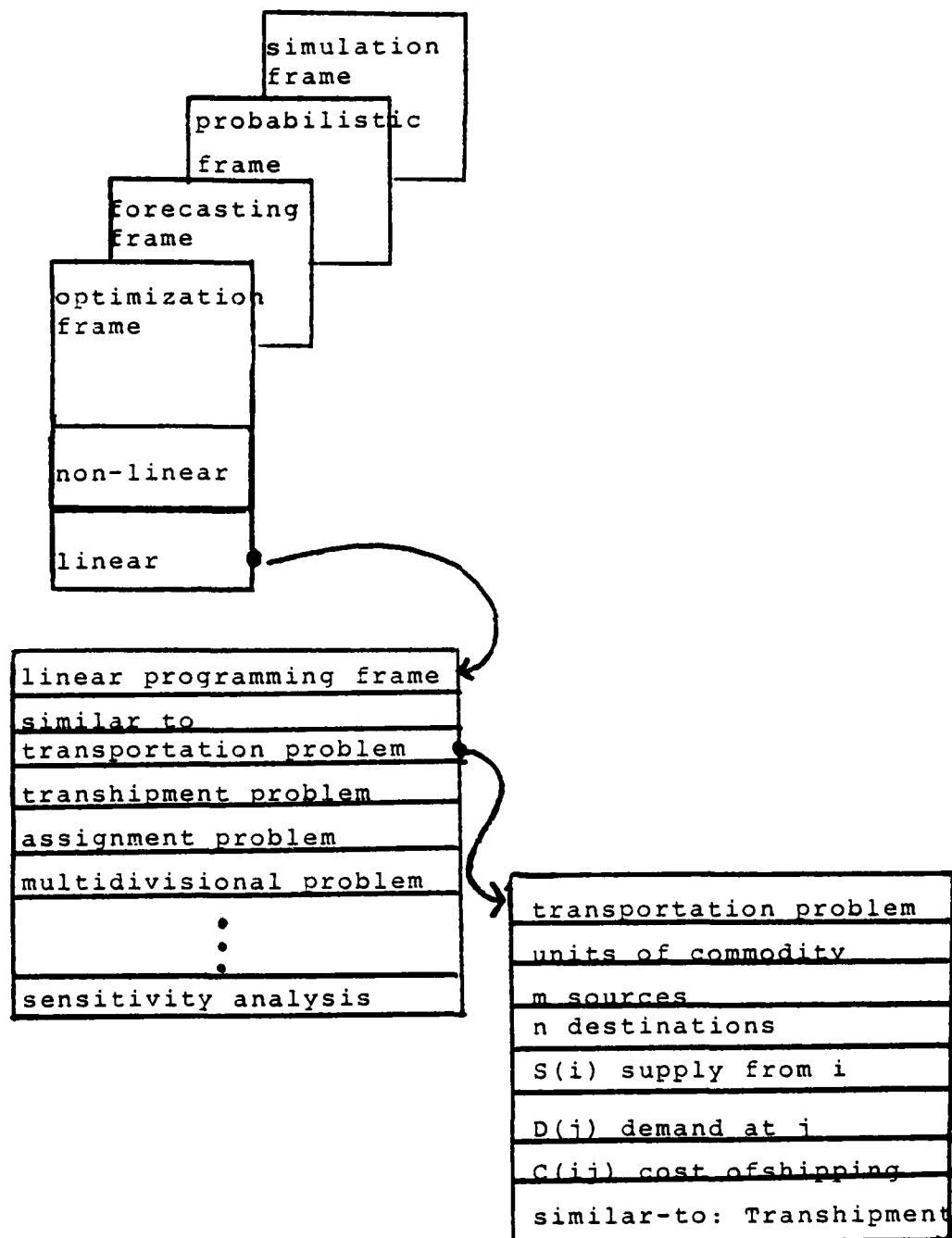


Figure 4.5: A Model Base of Frames

describe at least two, and these instances would be "facets" associated with the frame name of Transportation Problems. Facets are the values that a slot might assume. For example, when talking about a restaurant that we have not seen, we might picture what the tables look like. We have a mental picture of booths or tables. Therefore, this picture becomes a default value for the slot tables. In the frame, this value is kept in the default facet. The value facet will contain the actual value of the slot. If in reality our restaurant had only stools and a bar, then the value facet in the tables slot would have the entry "bar". The value of the facet would be the actual model instance. Depending upon how the frame is constructed, there may be several values associated with a facet. For example [Ref. 20], the general form in LISP might be the frame in Figure 4.6.

In the process of trying to match a problem with a frame, the various slots, facets, and values must be matched. Some default values will be established, and the trail eventually should lead to the correct (or most similar) existing frame. It is during this process that a "frame gestalt" is developed [Ref. 21]. A set of frames called the gestalt is selected, based on clues from input or gathered during the search. By examining each similarity and difference between frames, the system minimizes the possibility of missing obvious alternatives.

Frames show promise in settings that essentially are information systems. Many frames linked in a knowledge

```

((FRAME NAME) ((SLOT) ((FACET 1) (VALUE1)
                             (VALUE2)
                             .
                             .
                             .)
               (FACET 2) (VALUE1) . . .)
               .
               .
               .)
(SLOT2) (FACET1)...)...)
.
.
.)

```

Figure 4.6: The General Structure of a Frame [Ref. 20].

base constitute a "frame system". This frame system provides for the creation, modification, deletion, storage, and retrieval of knowledge from frames or related frames. The frame structure is used for pattern matching purposes during the execution of an application. The best example is a LISP program called DOCTOR [Ref. 22]. The user inputs a sentence, and the system matches it to a frame, looking for appropriate responses within that frame. A sentence entered, such as, "I am worried about my sister", evokes the response from DOCTOR, "Tell me about your sister".

In summary, representation of the model base in the frame format provides for establishing relationships among alternatives at the same level, and hierarchial relationships between model descriptions at different levels. In the implementation of frames, knowledge about models is included in the frame description. Considerations about both how much and what kind of knowledge to be included is yet to be discussed.

D. FRAMES AND ABSTRACTIONS

Turning our attention to a detailed look at the model abstraction offered by Dolk [Ref. 16], the idea of a model abstraction has its roots in data base management. It is convenient to conceptualize model base management in the same way we have learned to think about data base management. This is especially apparent when attempting to assemble data elements for an application in much the same

way we desire to assemble models. The same functions applicable in data base management are appropriate in model base management:

- (1) adding elements to a model,
- (2) deleting elements from a model,
- (3) creating elements,
- (4) modifying elements of a model.

Figure 4.7 illustrates the general structure of a model abstraction. It is clear that these procedures are important to satisfy certain imperative characteristics of the DSS, namely flexibility, adaptability, and user friendliness [Ref. 12].

Flexibility is achieved because the model abstraction has the ability to adjust to changing circumstances. Adaptability requires that new models be built in a trouble free manner, which is also characteristic of the model abstraction. Together these assets provide for a meaningful dimension of user friendliness.

Using Dolk's abstraction for the Simultaneous Equation Estimation Model (SEEM), we can take a detailed look at an abstraction. An abstraction has three elements: (1) data objects, (2) procedures, and (3), a set of assertions. The procedures of an abstraction act upon the specific data objects. Procedures such as creating an econometric model can be found in Dolk's abstraction for SEEM. Other procedures found there include deleting and adding equations

DATA OBJECTS

object 1

.

.

.

object i

PROCEDURES

procedure 1

.

.

.

procedure j

ASSERTIONS

assertion 1

.

.

.

assertion k

Figure 4.7: The General Structure of an Abstraction
[Ref. 16].

to the model. These features support the requirement for adaptability and flexibility so important to the effectiveness of the DSS.

Data objects are those elements acted upon by the operations. In SEEM, a data object is an equation which might be recursive or simultaneous. The procedures add and/or delete such equations to tailor the model to the circumstances. Data objects are the essential building blocks of the model, whereas procedures are the methods by which the model is constructed.

Assertions describe relationships between data objects and procedures. For example, an assertion in the SEEM model is that equations are either simultaneous or recursive. Assertions are important to ensure the proper development and/or integrity of the model.

Some important similarities and differences exist between model abstractions and the notion of frames. According to Dolk, model abstractions form a knowledge base for Model Management Systems (MMS). In addition, they are viewed as templates for constructing model instances. Facilities for doing this are not available, however, therefore a more flexible structure, the frame, is being considered. The model abstraction might be seen as a frame with slots and facets for procedures, data objects, and assertions. Abstractions, however, appear to be well-suited for manipulating model elements internally. Frames, on the

other hand, have the strengths of inheriting information external to the model. A frame system containing such knowledge as the class(es) of models could be used to construct dynamically new frames and model abstractions from existing ones. This would add deeper dimensions of adaptability and flexibility in the model base management system, which model abstractions cannot provide. The next Chapter outlines the specifications for such a system.

V. ORGANIZATION OF KNOWLEDGE ABOUT THE MODEL BASE

The question to be addressed in this Chapter is "how can frames augment model abstractions to form a more complete knowledge representation mechanism?" Each method has its relative strengths and weaknesses, but how can we combine the representation schemes so the strengths of each are retained? For this answer, we must reexamine the capabilities of a model management system in context of the frame and abstraction.

A. LEVELS OF ABSTRACTION

It is now clear, through the discussion on frames, that they are well-suited for hierarchical organization of knowledge. Frames are capable of representing several levels of abstraction quite easily. Model abstractions, on the other hand, appear best suited for first and second level abstractions of the model base. A first level abstraction is the lowest level of knowledge about a model. Lowest level means that it has the most detailed knowledge about a specific model. A second or higher level abstraction contains information that can be generalized to more than one model.

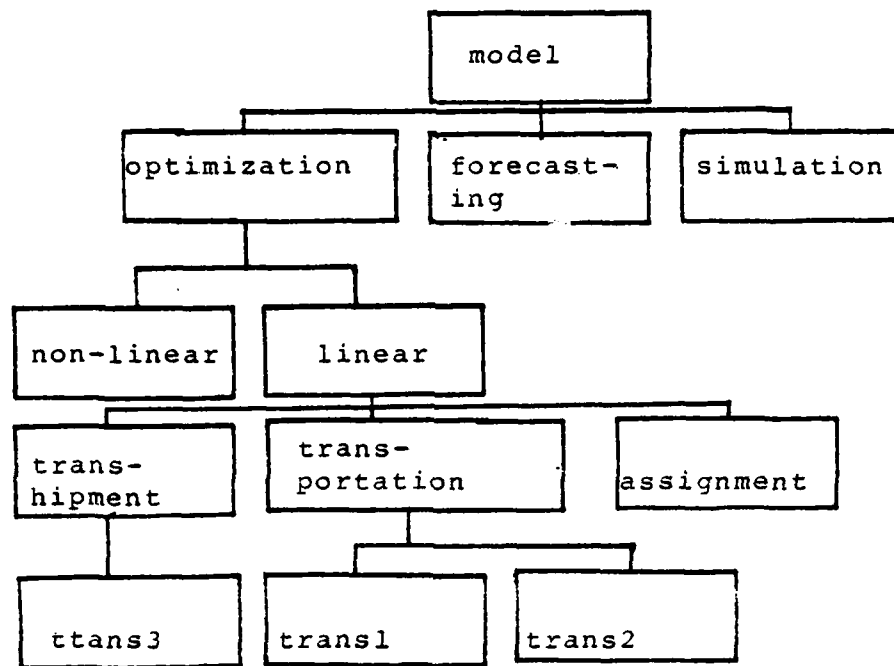
The concept of levels of abstraction is important in satisfying the objectives of a DSS outlined previously. Since a DSS must support a variety of decisionmaking processes easily, the system must be able to respond to

users with varying levels of knowledge and expertise about the model base.

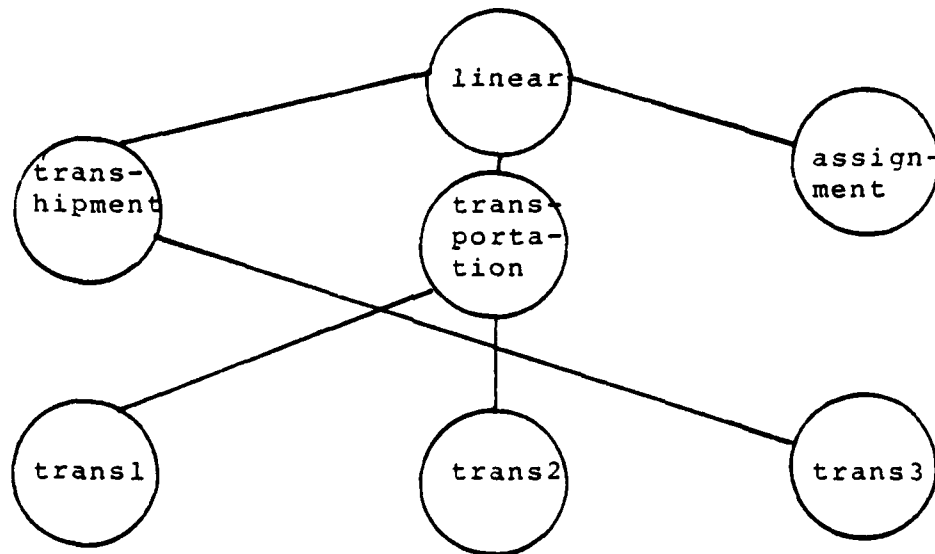
Bonczek, et. al. [Ref. 15] illustrate the notion of abstractions as applied to a knowledge base. Record types and record occurrences are central to their theme. A record type is the highest level abstraction, if it is not an occurrence of another record type. For example, the record for "models" is a record type. Instantiation of this record type includes record occurrences of optimization, forecasting, simulation, and probability. These instances are at one subordinate level beneath model. Optimization, for example, is both a record type and a record occurrence of models. Levels beneath optimization include linear and non-linear models. These also are record types, as well as occurrences, because linear programming models have such instances as transportation, assignment, and transshipment, among others. Even these are not on the lowest level, however, because actual model applications are seen as instances of transportation and other algorithms. Figure 5.1 illustrates the logical and occurrence structure of these levels of abstraction.

A model record, throughout levels two through five, is determined by the problem it solves. It can differ from similar records either by processing method, approach to the solution, or input requirements.

Furthermore, construction and execution of a desired model is based on modifying and combining various known



A typical logical structure



A typical occurrence structure

Figure 5.1: Logical and Occurrence Structures of a Model Base.

information-processing roles at all levels of the hierarchy. It is this aspect that endows the model base with the ability to be responsive to users with varying levels of knowledge about models. Knowledgeable model users can enter the knowledge base at lower levels of the hierarchy, whereas novice users can enter at higher levels. Therefore, the more levels of abstraction that characterize the system, the more generalized it becomes, and the less knowledge about the model base is demanded of the users.

Frames are ideal for representing knowledge on several levels of abstraction because of their ability to inherit from associated concepts and skills. On the other hand, model abstractions have the ability, through the use of assertions, to link to the data base in an orderly and efficient manner. In addition, the use of model abstractions has the added advantage of convenience. It is easier to think of the model base in the same terms as one thinks of the data base. In fact, model abstractions by definition and structure, are data base compatible. They are, however, limited in their ability to represent knowledge. Therefore, there exists the need to augment them with the characteristics of the frames. The next subsection compares these methodologies in detail, and proposes a means for combining them.

B. COMPARING FRAMES AND ABSTRACTIONS

Attention is next turned to identifying exactly which functions of the Model Base Management System are best supported by either models abstractions, or frame systems. It has been previously asserted that a MBSB must create new models quickly and easily. Because frame hierarchies require less expert knowledge from the model builder, frame systems more easily can create new models, and knowledge about those models, than can model abstractions. We have asserted that model abstractions are at the second level of abstraction. This is, however, only true if these abstractions identify their terminals at the first level. This is assumed and easily accomplished. Even at the second level, though, an intelligent user must somehow select and sequence processes for the model instance. Ideally, this selection and sequencing process would be automatic given the problem domain. Frames are well-suited for this process of integration because of their ability to reflect associations, inheritance, and similarities. This knowledge assists the model builder, whether manual or automated, to determine where the building blocks fit.

Model sequencing involves ensuring that models are executed in the proper order. Frames can assist in this process because of their hierarchical structure. For example, once a model frame is created which has more than one model building block, information about all the building

blocks becomes a part of the knowledge base, including information about when building blocks are executed.

Accessing existing models and model building blocks is another vital dimension of a model management system. This aspect also is better supported by the hierarchical structure inherent in frames. The access method may be quite general using frame linkages from any level in the hierarchy. Identification of optimization models may be done quickly without specific model names or structures. This is not apparent in model abstraction.

Conversely, model abstractions appear better suited for cataloging models. Information about procedures, data operators, and assertions for each model are important when manually evaluating or otherwise auditing the model base. The convenience of recording model base information in the same format as data base information contributes to clarity, economy, and consistency. In comparison, short of selectively reformatting knowledge, much of the knowledge contained within a frame would make little sense to a human user.

Model abstractions appear more appropriate in linking the model base to the data base. This primarily is due to their ability to validate how objects are structured and how procedures act upon those objects. These abilities are not inherent in frames, and would have to be builtin.

Model abstractions appear better able to manipulate a specific model once it is accessed through a frame scheme.

Model abstractions accommodate modifications for a specific model instance. As each model instance will have subtle differences, this ability is vital.

The development of models involves bringing together different classes of models. Some models might require both economic forecasting and linear programming models. Because of the frame's ability to hold knowledge about different classes of models, a frame system appears better suited for model creation. Figure 5.2 summarizes these comments comparing frames and abstractions.

C. COMBINING FRAMES AND ABSTRACTIONS

Now that the advantages of frames vis-a-vis abstractions have been compared, the next issue to resolve is which features to extract. Additionally, a decision must be made about the abilities a frame system should have, and also what specific information should be included in this system?

The abstraction concept presented by Dolk [Ref. 16] remains virtually intact. The structure of the abstraction will change slightly to fit into the frame system to be developed in subsection E. The use of the abstraction will alter conceptually as well. Third level abstractions essentially will become frames with slots and facets holding the information that the third level abstraction previously held. The lower the level of abstraction, the more specific the information becomes, thus fewer changes are

CHARACTERISTIC	FRAMES	ABSTRACTIONS
MODEL SEQUENCING	SYSTEM ASSISTED	USER DEPENDENT
MODEL ACCESSING	SYSTEM ASSISTED	USER DEPENDENT
MODEL CATALOGING	NONE	MODELS WELL DEFINED AND CATALOGED
MODEL LINKAGE TO DATA BASE	USER DEPENDENT	SYSTEM ASSISTED
MODEL MANIPULATION	USER DEPENDENT	SYSTEM ASSISTED
MODEL CREATION	LESS EXPERT KNOWLEDGE REQUIRED OF USER	USER DEPENDENT

Figure 5.2: Comparing Frames and Abstractions.

required to the abstraction. In other words, the essential aspects (such as data objects, procedures, and assertions) play a vital role in the frame system. Data objects still provide knowledge of what elements of the model are manipulated by the procedures. Assertions still will state characteristics of the specific model, such as linear objective functions and constraints. Finally, procedures still will identify those operators that will satisfy the objectives of the MBMS by additions, modifications, etc., to that model.

In general, a frame system will perform a number of functions [Ref. 20]:

- (1) once a frame, slot, and facet have been provided, the frame system will fetch information from the frame;
- (2) it can put information in the appropriate frame, slot, and facet location;
- (3) it has the ability to remove information within the frame, or remove the entire frame;
- (4) if no special information is available for a model, default values should be accessed;
- (5) inheritance from parent or otherwise associated frames should occur; and,
- (6) inheritance for default values also should occur.

All that remains is to generalize about the information a frame system manipulating model knowledge should have. This is dependent largely upon the level of sophistication

anticipated of the users, and the complexity of the application. Lenat [Ref. 23] suggests that in addition to procedures, data objects, and assertions about a specific model, we would like to know:

(1) Generalizations--Which other models have less restrictive requirements (facilitate nonlinear constraints for example)? In essence, we move up the hierarchy of constraints with this information.

(2) Specializations--Which models are specialized applications? For example, the transportation model is a specialization of the linear programming model. We will call these specializations. Here we move down the hierarchy, adding constraints to our model formulations.

(3) Examples--For what kinds of situations is the model appropriate?

(4) Is-a--What kind of model is it (e.g., optimization, forecasting, etc.)? These will be identified through the is-a link.

(5) Views--How can a view of the specific model in the context of another be obtained? For example, how is integer programming similar to other linear programming models, and how is it different? This might be called information views.

(6) Similarities--What other model(s) is the current instance similar to?

(7) Definitions--How can it be determined if the circumstance at hand will fit a model in the model base?

(8) Preparations--How can the input model be prepared?
This can be called "preparations".

(9) Integration--Can this model be integrated with
other models?

(10) Execution--How should this model be run?

Figure 5.3 is an example of how this abstraction would
look when combined with knowledge about frames.

We now are prepared to develop a simple example of how
all this information can be resident in a knowledge base of
frame representation. The next section develops such an
implementation.

D. THE MODEL DOMAIN

The knowledge base presented here makes use of typical
examples of models used in a business decisionmaking
environment. This knowledge is organized into a frame
system. Frames intuitively are pleasing for this task.
Problem solvers normally have a frame of reference that
includes the parameters of the problem. For example, a
distribution system manager may have a problem involving
sources of product, destinations, and cost minimization.
It is the goal of the frame system to accept what the user
knows about a problem, and to translate it into knowledge
about a model instance. This is a matching process, i.e.,
an effort to bring the problem and the model together
through an interpretation of the parameters.

Data Objects

object 1

.

.

object i

Procedures

procedure 1

.

.

procedure j

Assertions

assertion 1

.

.

assertion k

Generalizations

Specializations

Examples

Is-a

Views

Similarities

Definitions

Preparations

Integration

Execution

AKO

Instances

Figure 5.3: Convention for Describing a Frame-like Abstraction.

The concept of hierarchy is important to facilitate the notion of inheritance. Frame representations are ideal here for modeling work. As the problem solver clarifies the issues involved, there is a movement from the general, or high level of abstraction, to the specific, or lower levels of abstraction--possibly to an actual model instance.

In our example, the world is subdivided into two major categories, shown in Figure 5.4; these categories are optimization and forecasting. This is, of course, a simplification for the purpose of demonstrating the use of frames. Optimization problems are composed of linear programming and nonlinear programming models. We do not expand the nodes of forecasting nor nonlinear programming. Linear programming is composed of the transportation model and the assignment model. Finally, the transportation model is subdivided into actual instances of the algorithm, namely trans1, trans2, and trans3.

Figures 5.5 and 5.6 are examples of the frames contained in the knowledge base. The level of the frame in the hierarchy determines the level of detail in the frame. Figure 5.5 is a first level instantiation of trans1. The detailed knowledge about trans1 is extensive. Figure 5.6 is a fourth level frame; the knowledge contained in the "optimization" frame is relatively general and limited. The concept of general to specific is common among decision-makers as they attempt to match parameters of their problem instance to a solution method.

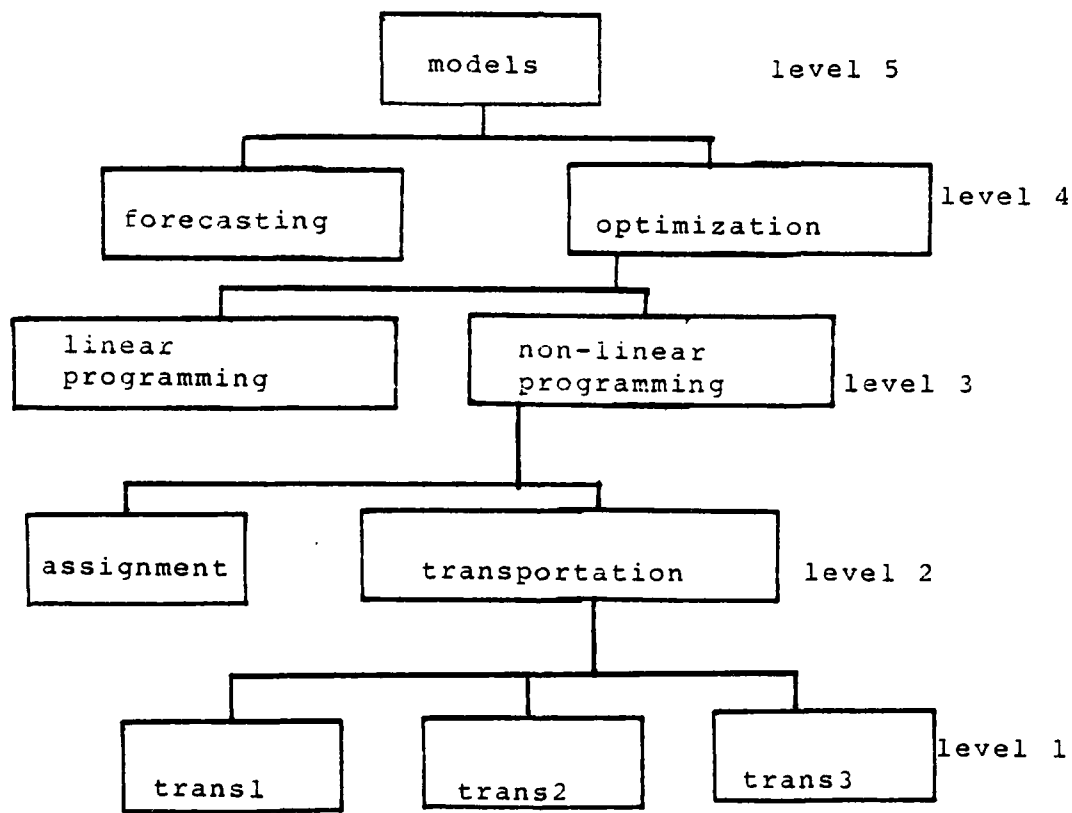


Figure 5.4: Organization of the Frame System.

```
(transl  (features (value (adjective)))
         (ako (value (transportation)))
         (generalizations (default (linear-programming)))
         (specializations)
         (example (value (sources and destinations)))
         (views (value (northwest-corner, stepping-stone)))
         (analogies (value (trans 1 trans2)))
         (definitions (value (cost of shipments)))
         (input (value (script for input)))
         (sequencing (value (script for sequencing)))
         (execution (value (script for execution))))
```

Figure 5.5: Example of a Lower Level Frame.

```
(optimization (features (value adjective)))  
  (adjective-function (value (optimize)))  
  (ako (value (model)))  
  (instance (value (lin-prog non-lin-prog))))
```

Figure 5.6: Example of a Higher Level Frame

Finally, in our model world, user interaction is required as we migrate from one level to the next. The user must specify which node on the next level needs to be expanded. A demonstration of a simple frame system follows.

E. FRAME SYSTEM FUNCTIONS

The frame system implemented here is an adaptation of Winston and Horn's [Ref. 20] frame system to a model knowledge base. There are 15 programs in this system, the major features of which are demonstrated below. They are written in LISP, and are contained in Appendix A.

The simple frame system developed here is unlike the abstractions developed by Dolk [Ref. 16] in one important regard: this frame system does not currently include predicate calculus statements. Because predicate calculus is not incorporated, the present system is not able to make deterministic inferences about models and problems. Predicate calculus would be straightforward to implement within the facets of the frame structure. The following paragraphs describe a basic frame system capable of expansion to a full-knowledge representation scheme.

The program FGET retrieves information from a frame if it is given a frame, slot, and facet. It is worthwhile to recall that a frame possesses a name, some general features which are slots, and some words describing those features which are facets. Thus: (fget'transl 'ako 'value) returns transportation.

Conversely, FPUT places information in a frame given the frame, slot and facet: (fput 'trans2 'ako 'value 'transportation) returns (transportation).

If this information is later learned to be incorrect, REMOVE will erase it: (remove 'trans2 'ako 'value 'transportation) returns (t).

The "ako" element above stands for the phrase "a kind of". The ako slot is very useful for inheritance. All transportation models have certain things in common. These common characteristics would appear under the frame name of "transportation". Instances of transportation models, such as trans2, would inherit these elements. For example, trans2 would inherit the element of linear programming.

Frame functions may look in several facets to satisfy a request. FGET-V-D looks first in the "value" facet, and then in the "default" facet, in an attempt to retrieve information: (fget-v-d 'trans1 'generalizations) returns (linear-programming).

Recall that the "value" facet holds the actual value for the specific slot being described. In the restaurant example, the tables slot might have had the value "French Provincial", and the default may have simply been "booths".

In addition, if the frame system fails at finding information in the value or default facets of the slot, it may look for something in the "if-needed" facet; this is the role of FGET-V-D-F. The if-needed facet contains

procedures to be executed if the value and default slots have no values in them. In this case, the system searches first for a value and then a default, and failing at these, it looks in the if-needed facet. It finds in this slot a "demon"--a function that is executed when nothing is to be found in the value or default facets:

```
(fget-v-d-f 'transl 'specialization)
returns (please supply a value for the specialization
        slot in the transl frame.)
>'transshipment
returns (transshipment).
```

A function called FGETCLASSES returns the name of all frames that a given frame is related to through the ako slot. In this case, the ako slot acts as a link to higher generalizations about the transl frame:

```
(fgetclasses 'transl)
returns (transl transportation (linear-programming)).
```

This is very useful for inheritance among frames that are linked through the ako slot.

FGET-I is a program that looks for information in a related frame if it finds nothing in the current frame. In the following example we find nothing in the transl specialization slot, so we look for it and find it in the specialization slot of the transportation frame. In this way subordinate (or child frames) inherit information from senior (or parent) frames: (fget-i 'transl 'specializations)

```
returns (transshipment).
```

Finally, demons may be activated if we add a particular facet to a frame. Say, for example, that we add an instance to a frame, it stands to reason that a frame should be created for each instance. We put these types of demons in the if-added facet of the slot in the frame associated through an ako link:

```
(fput+ 'transl 'specialization 'default 'linear-  
programming)
```

returns (procedures to be followed when adding specializations).

Two interesting features of the frame system are accomplished through the functions FCHECK and FCLAMP. FCHECK is a function that takes as input a frame name, slot, facet, and value, and checks to see if that value exists for the specified slot:

```
(fcheck 'transl 'ako 'value 'transportation)
```

returns (t) if true, and (nil) otherwise.

FCLAMP is a function that ties two frames together so that anything that goes into one frame will go into the other. It takes as input the two frame names and the slot that is to be duplicated between them. It works like this:

```
(fclamp frame1 frame2 slot)
```

Now anything that is entered in the specified slot will be entered automatically into the other frame. It may be an interesting incumbent of the if-added facet.

This summarizes the abilities of the implemented frame system. In the model domain, the strength of any system can be found in its matching and reasoning ability.

Suppose the user only knows that the problem at hand deals with minimizing the costs of shipping goods from some sources to destinations. The resulting candidate frame will consist of only a few slots: costs, goods, sources, and destinations. Armed with this information, the system tries to match this candidate frame to existing patterns in the knowledge base [Ref. 25]. A frame system has the ability to respond to this type of sketchy problem by answering several important questions:

- (1) Does this candidate match an existing frame?
- (2) If it does, is it an exact match?
- (3) If it is not an exact match, which elements would have to match to make it an exact match?
- (4) Are the unmatched elements so important that the pattern can not be used?
- (5) If the existing pattern can be used, how would the candidate elements be translated into the existing pattern's elements? (For example, how can we decode that goods are equivalent to commodities?)

A frame structured knowledge base with these reasoning capabilities makes for a robust system.

Once the appropriate model has been found, the user needs to know how to prepare input for it, and how to use

it (execute it). Here the idea of a script is introduced [Ref. 18]. A script generally is defined as a standardized sequence of events describing some activity in stereotypical fashion, such as eating food or visiting friends. The script is similar functionally to a frame, and thus fits neatly into the knowledge representation presented here. Most important, however, is its ability to anticipate certain events in an activity. Figure 5.7 is an example of a script. Scripts appear to be ideal for preparing input for a model, and for executing a model.

Input preparation might include such tasks as formatting, sequencing, editing, etc. The next section attempts to make it easier to use a frame system by attaching a simple English-like command language to it.

F. ENGLISH-LIKE COMMAND SYSTEM

One of the impressive strengths of the frame system is its ability to respond to English-like commands. Frame representation languages have been developed to take advantage of this strength [Ref. 14].

The front-end implemented here is a simple noun group parser which is explained more fully in the following section. It is written in LISP, and also is an adaptation of Winston and Horn's [Ref. 20] English compiler. The programs are listed in Appendix B, and a short User's Guide is enclosed in Appendix C.

```
(input-preparation (model (value (transl))
  (instructions (value (formulate tableau))
    (first (name rows and columns)
      (second (assure rhs's are positive))
      (third (specify rows + 1))
      (fourth (specify variables))
      (fifth (enter reader number))
      (sixth (enter printer number))
      (seventh (adjust dimensions))
      (eighth (adjust format statements))
      (ninth (call execution)))
```

Figure 5.7: An Example of an Input Script.

As input, our system receives a command verb such as "identify", and a list of adjectives clarifying a noun (see Figure 5.8). This results in an easily used system that is more friendly to the user than the frame system previously described.

For example, in the previous section, the command "to identify" the transportation models was:

```
(fget 'transportation 'instance 'value)
returning (transl trans2 trans3).
```

With English-like language we now can ask simply:

```
(request: identify the transportation models)
returning (there are) (transl trans2 trans3). In addition,
if the user wants to count the number of transportation
models in the model base, the command is given as:
```

```
(request: count the transportation models)
answering (there are) (3).
```

Of course there is a full range of commands required in model base management such as: add, delete, and modify. This command structure can be accomplished easily with the LISP programs contained in Appendix B, by modifying the parse-command program to "execute the frame programs previously identified".

This system works in four phases as illustrated in Figure 5.9. When the noun phrase is entered, the first event that occurs is the building of a parse-tree. The parse-tree contains such items of interest as the list of

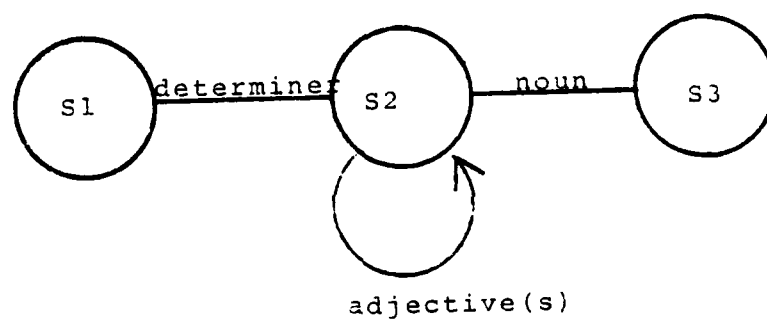


Figure 5.8: Schematic of an Augmented Transition Network.

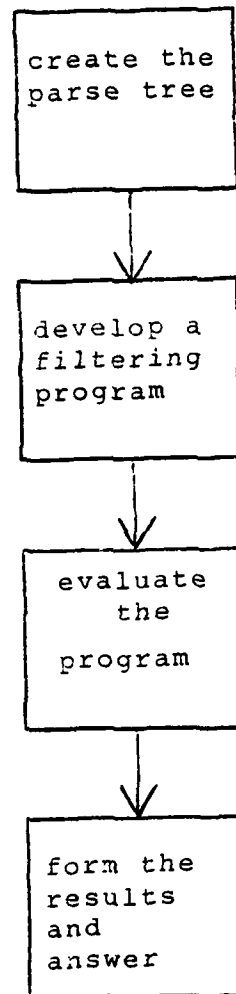


Figure 5.9: The Four Phases of Answering the English Query [Ref. 20].

adjectives, the noun (whether singular or plural), and whether the determiner was definite or indefinite, Figure 5.10. This tree is placed on a property list for the specific input phrase. The second step involves accessing these values and constructing a temporary LISP program from them. Third, this newly constructed program is executed. During the execution, it retrieves from the knowledge base (of frames) items meeting the qualifications identified by the adjectives of the input phrase. Fourth, the results of program execution are processed and presented to the user as a response to the input request.

Of course, one would want to do more than simply identify the models, or to know how many of a given model class of models exist. A full set of commands would include all of the manipulation parameters previously mentioned. For the model domain, this is a relatively compact vocabulary, dependent upon the degree of adaptability of English the builder wants to incorporate. For example, the system in Section Four can answer the question: "Identify the models". However, if the request is phrased "which are the models?" the system could not respond. While natural language interfaces can be difficult to implement, an English-like command vocabulary is a realistic objective with a high payoff in usability and flexibility when combined with frame system knowledge base.

Input: A typical sentence.
node: children: (A typical sentence
 determiner: indefinite number:
 singular adjective(s):
 typical noun:
 sentence.

Figure 5.10: General Structure of a Parse-tree (Ref. 20].

VI. CONCLUSIONS

A. AREAS FOR FURTHER RESEARCH

The purpose of this section is to summarize the research presented, and to present areas for further research. The first area for further research involves developing the English-like command system for a frame representation scheme on a model base of representative models. The overall objective is to develop a vocabulary to access, integrate, and manipulate both the model base and the data base. A second area of research would attempt to identify what specific knowledge is required in a comprehensive model base.

In our example we used frames, with an Augmented Transition Network (ATN) as a knowledge base only. That is, our English-like commands did not invoke the frame functions described in Chapter IV (E). In our system, the user was required not only to know how to manipulate frames with frame functions, but also to know an English-like command vocabulary. Therefore, in addition to developing all of the manipulation commands for model base management, our English-like system should, in the future, invoke the frame system functions as well.

Since we expect the user to become acquainted with our modeling language, it would be most convenient if those same commands interfaced with the data base. This raises many issues:

- (1) In which data structure should the data base be implemented? Are frames appropriate for typical data items? Do human factors such as response time outweigh the convenience of common English commands for the data base and model base?
- (2) If data are not organized the same way models are, how much system maintenance overhead will be incurred to integrate models and data?

Once we have developed basic English-like commands for our interface to the model base, usage becomes an issue. Are there some commands used in sequence so frequently as to warrant a macro-command to accommodate users? For example, is the command "identify" issued in sequence so often that a command such as "browse" is appropriate? Or, is the sequence "identify" and "update" so frequent that a single update command would both access and update a frame? Other usage statistics may involve performance such as response-time, or session-time, for a specific request. Still others may look at the size of the knowledge base relative to performance, and identify that knowledge rarely accessed for possible pruning.

Thus the field of natural language, and English-like command languages, have interesting challenges and significant payoffs.

B. IMPACT ON THE DSS PARADIGM

This study has focused on the problem of managing knowledge about models within a DSS. Consequently, the paradigm presented in Figure 2.4 differs only in regard to its modeling component. Figure 6.1 illustrates this change.

The dialog management system still must pass to the modeling component the appropriate commands and identifiers. In addition, the data base still must pass to the model base the information required to execute the models. Furthermore, the presentation of model results remains unchanged.

The essential feature that has changed is the concept of a frame management as a model base management facility. The frame management system is one of several front-ends that may have been applied to the problem. Production rules are another favorite among builders of expert systems. The frame management system was selected because of its adaptability and flexibility. It has, we believe, the most promise of model management schemes to date.

This paradigm thus becomes a more powerful configuration of Figure 2.4. The underlying reason for this improvement rests with the fact that the knowledge base now resides with the model manager, and not with the user. The value of this improvement is significant, as Keen [Ref. 2] helps to point out:

- (1) because it is easier to access models, the number of alternatives considered will increase;

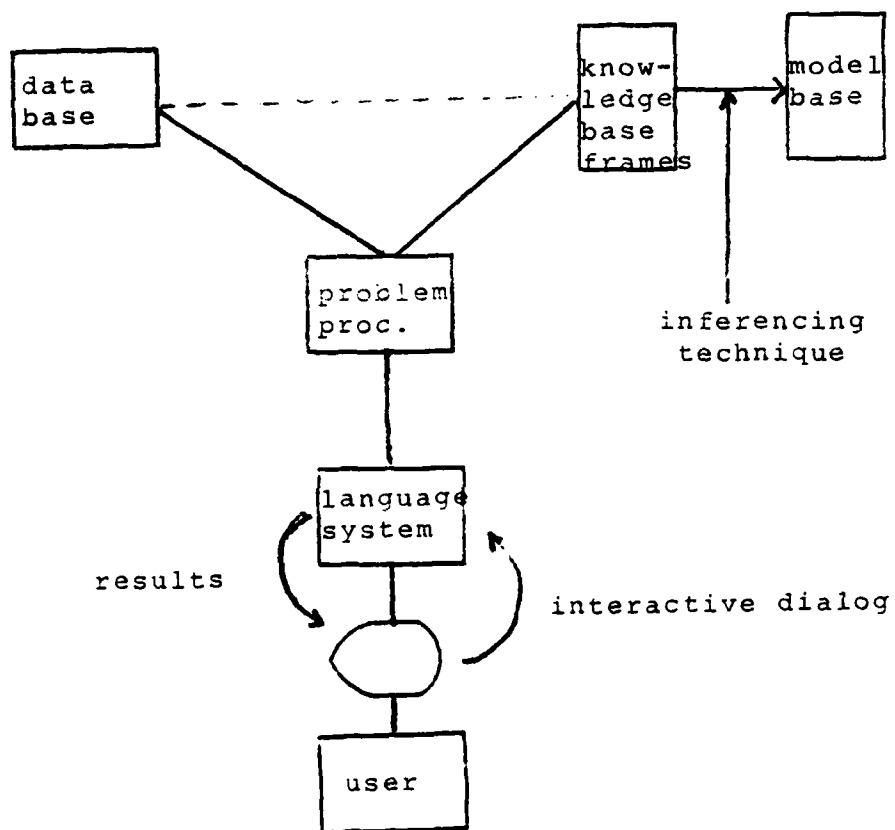


Figure 6.1: A DSS Paradigm with Knowledge Base Support.

- (2) because the frame configuration facilitates a variety of approaches to a situation, it is responsive to unexpected demands and situations;
- (3) there is a cost savings in clerical labor for the collection and massage of input data for manual modes of execution.
- (4) there is a time-savings realized because the time using the system is more effectively spent; and,
- (5) all of the above will increase the quality of decisions made, and the effectiveness of the resources consumed for the making of that decision.

C. SUMMARY

This investigation has made the following attempts:

- (1) to identify the myriad of requirements and challenges faced by the implementor of a DSS;
- (2) to identify the areas of AI that might be of value in modeling the decisionmaking domain of an organization;
- (3) to show how model abstractions can be extended to the concept of frames; and,
- (4) to implement a simplistic frame system as the basis of a model management scheme.

This study has been an attempt to bind together the fields of organizational behavior and computer science in areas divergent from traditional data processing. The convergence of Artificial Intelligence and Decision Support

Systems is a promising area of research. Some forms this research might take with regard to model management have been presented.

As Minsky said: "Thinking always begins with suggestive but imperfect plans and images; these are progressively replaced by better, but usually still imperfect, ideas."

APPENDIX A

PROGRAM LISTINGS FOR THE FRAME SYSTEM

[Ref. 20]

```
(defun fget (frame slot facet)
  (mapcar 'car
    (cdr (assoc facet
      (car (assoc slot
        (car (get frame "frame"))))))))
```

```
(defun fassoc (key a-list)
  (cond ((assoc key (car a-list))
    (t (car (rplacd (last a-list)
      (list (list key)))))))
```

```
(defun fgetframe (frame)
  (cond ((get frame "frame")
    (t (putprop frame (list frame) "frame"))))
```

```

(defun fput (frame slot facet value)
  (cond ((member value (fget frame slot facet)) nil)
        (t (fassoc value
                    (fassoc facet
                          (fassoc slot
                                (fgetframe frame))))
            value)))

```

```

(defun p (message)
  (print (squash message)))

```

```

(defun squash (s)
  (cond ((null s) nil)
        ((atom s) (list s))
        (t (append (squash (car s))
                    (squash (cdr s))))))

```

```

(defun builder ()
  (fput 'a 'features 'value '(determiner singular indefinite))
  (fput 'the 'features 'value '(determiner definite))
  (fput 'long 'adjective-function 'value 'longp)
  (fput 'long 'features 'value '(adjective))
  (fput 'red 'features 'value '(adjective))
  (fput 'large 'features 'value '(adjective))
  (fput 'large 'adjective-function 'value 'largep)
  (fput 'screwdrivers 'features 'value '(noun plural))
  (fput 'screwdriver 'instance 'value '(s1 s2 s3 s4 s5))
  (fput 'screwdrivers 'singular-form 'value 'screwdriver)
  (fput 'screwdrivers 'features 'value '(noun plural))
  (fput 's1 'size 'value 'large)
  (fput 's2 'size 'value 'large)
  (fput 's3 'size 'value 'large)
  (fput 'model 'features 'value '(noun singular))
  (fput 'model 'instance 'value '(optimization forecasting
                                linear-programming non-linear-programming
                                transportation assignment trans1 trans2 trans3))

  (fput 'models 'features 'value '(noun plural))
  (fput 'models 'singular-form 'value 'model)
  (fput 'optimization 'features 'value '(adjective))
  (fput 'optimization 'adjective-function 'value 'optimize)
  (fput 'optimization 'ako 'value 'model)
  (fput 'optimization 'instance 'value '(linear-programming
                                non-linear-programming))

```

```

(fput 'trans1 'features 'value '(adjective))
(fput 'trans1 'ako 'value '(transportation))
(fput 'trans1 'generalizations 'value '(linear-programming))
(fput 'trans1 'specializations 'if-needed 'ask)
(fput 'trans1 'example 'value '(sources and destinations))
(fput 'trans1 'views 'value '(north-west corner))
(fput 'trans1 'analogies 'value '(trans2 trans3))
(fput 'trans1 'definitions 'value '(costs of shipment))
(fput 'trans1 'input 'value '(script for input))
(fput 'trans1 'sequencing 'value '(script for sequencing))
(fput 'trans1 'execution 'value '(script for execution))
(fput 'transportation 'features 'value '(adjective))
(fput 'transportation 'ako 'value '(linear-programming))
(fput 'transportation 'specializations 'value '(transshipment))
(fput 'transportation 'specializations 'if-needed 'ask)
(fput 'transportation 'instance 'value '(trans1 trans2 trans3))
(fput 'transportation 'adjective-function 'value '(transport))
(fput 'forecasting 'ako 'value '(model))
(fput 'linear-programming 'features 'value '(adjective))
(fput 'linear-programming 'features 'value '(optimization))
(fput 'linear-programming 'adjective-function 'value '(linprog))
(fput 'linear-programming 'instance 'value '(transportation assignment))
(fput 'non-linear-programming 'ako 'value '(optimization))
(fput 'assignment 'ako 'value '(linear-programming))
(fput 'trans3 'ako 'value '(transportation))
(fput 'trans2 'features 'value '(adjective))
(fput 'trans2 'ako 'value '(transportation))
(fput 'trans2 'specialization 'value '(transshipment))

```

```

(defun fremove (frame slot facet value)
  (prog (slots facets values target)
    (setq slots (fcetframe frame))
    (setq facets (assoc slot (cdr slots)))
    (setq values (assoc facet (cdr facets)))
    (setq target (assoc value (cdr values)))
    (delete target values)
    (cond ((null (cdr values))
           (delete values facets)))
    (cond ((null (cdr facets))
           (delete facets slots)))
    (return (not (null target)))))

```

```

(defun fcheck (frame slot facet value)
  (cond ((member value (fcet frame slot facet)) t)
        (t nil)))

```

AD-A132 211

KNOWLEDGE BASE MANAGEMENT FOR MODEL MANAGEMENT SYSTEMS
(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA G W WATSON
JUN 83

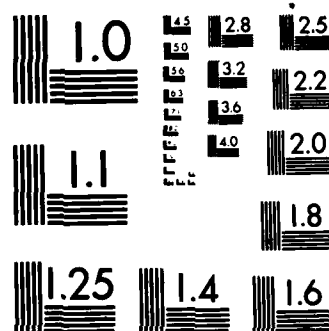
2/2

UNCLASSIFIED

F/G 5/1

NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

```

(defun fclamp (frame1 frame2 slot)
  (rplacd (fassoc slot (fgetframe frame1))
    (cdr (fassoc slot (fgetframe frame2))))
  slot)
(defun fget-v-d (frame slot)
  (cond ((fget frame slot 'value))
    ((fget frame slot 'default))))

(defun fget-v-d-t (frame slot)
  (cond ((fget frame slot 'value))
    ((fget frame slot 'default))
    (t (mappcar 'funcall
      (fget frame slot 'if-needed)))))

(defun ask ()
  (print (append "(please supply a list for the)"
    (list slot)
    "(slot in the)"
    (list frame)
    "(frame)"))
  (read))

(defun fget-1 (frame slot)
  (prog (classes result)
    (setq classes (fgetclasses frame))
    loop
    (cond ((null classes) (return nil))
      ((setq result
        (fget (car classes) slot 'value))
        (return result))
      (t (setq classes (car classes))
        (go loop)))))

(defun fput+ (frame slot facet value)
  (cond ((fput frame slot facet value)
    (mappcar '(lambda ()
      (mappcar 'funcall
        (fget slot 'if-added)))
      (fgetclasses frame))
    value)))

```

```

(defun fgetclasses (frame)
  (prog (queue progeny classes)
    (setq queue (list frame))
    tryagain
    (cond ((null queue) (return (reverse classes)))
          ((not (member (car queue) classes))
           (setq classes (cons (car queue) classes))))
    (setq progeny (fget (car queue) 'ako 'value))
    (setq queue (cdr queue))
    (setq queue (append queue progeny))
    (go tryagain)))

```

```

(defun fget-z (frame slot)
  (prog (classes result)
    (setq classes (fgetclasses frame))
  loop
    (cond ((null classes) (return nil))
          ((setq result
                  (or (fget-v-d (car classes) slot)
                      (mapcan '(lambda (e) (apply e nil))
                              (fget (car classes) slot 'if-needed))))
           (return result))
          (t (setq classes (cdr classes))
              (go loop)))))

```

```

(defun fremove+ (frame slot facet value)
  (cond ((fremove frame slot facet value)
        (mapc '(lambda (e)
                  (mapc '(lambda (f) (apply f nil))
                        (fget e slot 'if-needed)))
              (fgetclasses frame))
        value)))

```

```

(defun fget-n (frame slot)
  (prog (classes result)
    (setq classes (fgetclasses frame))
  loop1
    (cond ((null classes)
          (setq classes (fgetclasses frame))
          (go loop2))
          ((setq result (fget (car classes) slot 'value))
           (return result)))

```

```

      (t (setq classes (car classes))
        (go loop1)))
loop2
(cond ((null classes)
      (setq classes (fgetclasses frame))
      (go loop3))
      ((setq result (fget (car classes) slot 'default))
       (return result))
      (t (setq classes (cdr classes))
          (go loop2)))
loop3
(cond ((null classes) (return nil))
      ((setq result (mapcan "(lambda (e) (apply e nil))
                            (fget (car classes)
                                slot
                                'if-needed)))
       (return result))
      (t (setq classes (car classes))
          (go loop3)))))

```

APPENDIX B

PROGRAM LISTINGS FOR ENGLISH-LIKE COMMANDS

[Ref. 20]

```
(defun request: fexpr (remaining-words)
  (prog (tree program results noun-group)
    (setq tree (parse-command (gensym) nil))
    (setq program (make-search-program (get tree 'noun-group)))
    (setq results (eval program))
    (cond ((equal (get tree 'command) 'count)
      (cond ((car results) (p "(there are)"))
            ((length results))
            (results (p "(there is)"))
            (o 1))
      (t (p "(sorry the request and data base clash)"))))
    ((equal (get tree 'command) 'enumerate)
      (cond ((cdr results) (p "(the results are:)"))
            ((p results))
            (results (p "(the result is:)"))
            ((p results))
            (t (p "(sorry the request and the data base clash)")))))
```

```
(defun compile macro (description)
  (prog (name body program beginning middle end)
    (setq name (cadr description))
    (setq body (cadr description))
    (setq beginning
      (subst name
        'replace
        '(prog (this-node hold)
          (setq hold remaining-words)
          (setq current-word (car remaining-words))
          (setq this-node (gensym 'replace))))))
    (setq
      middle
      (apply
        'append
        (mapcar
          '(lambda
            (state)
              (list (car state)
                (cons 'cons
                  (append
                    (mapcar
```

```

      (lambda
        (clause)
        (append (list (cadr clause))
                 (cons ((caddr clause)
                       (car (caddr clause)))
                       (list (list 'go (caddr clause))))
                 (cdr state))
        '(((t (go lose))))))
    body)))
(setq end
  '(win (cond ((not (testi this-node features)) (go lose)))
        (attach this-node parent-node)
        (setq last-parsed this-node)
        (return this-node)
        lose
        (setq remaining-words hold)
        (setq current-word (car remaining-words))
        (return nil)))
(setq program (append beginning middle end))
(return (list 'defun name '(parent-node features) program)))

```

```

(defun make-search-program (node)
  (append
    (subst (prog (noun)
      (setq noun (get node 'noun))
      (return (cond ((member 'plural (car (iget noun 'features 'value)))
                    (car (iget noun 'singular-form 'value)))
                    (t noun))))
      'noun
      '(prog (objects)
        (setq objects (car (iget 'noun 'instance 'value))))))
    (mapcar
      '(lambda (adjective)
        (subst (car (iget adjective 'adjective-function 'value))
          'predicate
          '(seto objects
            (mapcar
              '(lambda (candidate)
                (cond ((predicate candidate)
                      (list candidate))
                      (t nil)))
              objects))))
        (get node 'adjectives))
      (subst (prog (determiner number)
        (seti determiner (get node 'determiner))
        (seto number (get node 'number))
        (return
          (cond ((equal determiner 'definite)
                (cond ((equal number 'singular)
                      '(equal (length objects) 1))
                      ((equal number 'plural)
                       '(greaterp (length objects) 1))))
          (t nil))))
        (get node 'determiners))
      (subst (prog (number)
        (seti number (get node 'number))
        (return
          (cond ((equal number 'singular)
                (equal (length objects) 1))
                ((equal number 'plural)
                 '(greaterp (length objects) 1))))
          (t nil))))
        (get node 'numbers))
    (get node 'program)))

```

```

((equal determiner 'indefinite)
 (cond ((equal number 'singular)
        '(greaterp (length objects) 0))))
((number number)
 (list 'greaterp '(length objects) number))))
'test
'((cond (test (return objects))
        (t (return nil))))))

```

```

(compile parse-command
  (s1 (if (and (equal current-word 'count)
              (parse-word this-node '()))
        -> s2
        after
        (setf 'command 'count))
    (if (and (equal current-word 'identify)
            (parse-word this-node '()))
        -> s2
        after
        (setf 'command 'enumerate)))
  (s2 (if (and (parse-noun-group this-node nil)
              (null remaining-words))
        -> win
        after
        (setf 'noun-group last-parsed))))

```

```

(compile parse-noun-groupx
  (s1 (if (parse-word this-node 'determiner)
        -> s2a
        after
        (setf 'number (select '(singular plural)
                              (getf last-parsed)))
        (setf 'determiner (select '(definite indefinite)
                                   (getf last-parsed))))
    (if t -> s2a))
  (s2a (if (parse-word this-node 'number)
          -> s2
          after
          (cond ((equal 'singular (getf 'number))
                 (print 'tilt-determiner-number)))
          (setf 'number (get last-parsed 'number)))
    (if t -> s2))
  (s2 (if (and remaining-words
              (select '(adjective-noun)
                      (getf (cdr remaining-words)))
              (parse-word this-node 'adjective))
        -> s2
        after

```

```

(addr 'adjectives last-parsed))

(if (parse-word this-node 'noun)
    => win
    after
    (cond ((equal
            (length (intersection '(singular plural)
                                   (cons (getr 'number)
                                         (getf last-parsed))))
            2)
            (print 'tilt-determiner-noun))
          ((and (numberp (getr 'number))
                 (member 'singular (getf last-parsed)))
            (print 'tilt-number-noun))
          (cond ((not (numberp (getr 'number)))
                  (setr 'number (select '(singular plural)
                                          (getf last-parsed))))))
    (setr 'noun last-parsed)))

(compile parse-word
  (si (if t => win
        after
        (setc this-node current-word)
        (setc remaining-words (car remaining-words))
        (cond (remaining-words (setc current-word
                                       (car remaining-words)))
              (t (setc current-word nil))))))

(defun setr (register value)
  (putprop this-node value register)
  value)

(defun getr (register)
  (get this-node register))

(defun select (x y)
  (cond ((null x) nil)
        ((member (car x) y) (car x))
        (t (select (car x) y))))

```

```

(defun genname (name)
  (prog (n)
    (cond ((setu n (get name 'namecounter)))
          (t (setu n 1)))
    (putprop name (add1 n) 'namecounter)
    (return (implode (append (explode name)
                              (explode n))))))

```

```

(defun largep (object) (equal (car (fget object 'size 'value)) 'large))

```

```

(defun redp (object) (equal (get object 'color) 'red))

```

```

(defun longp (object) (greaterp (or (get object 'length) 0.) 5.0))

```

```

(defun getf (x) (car (fget x 'features 'value)))

```

```

(defun testf (node features)
  (cond ((null features))
        ((atom features)
         (setu features (list features))))
  (equal (length features)
         (length (intersection features (getf node)))))

```

```

(defun attach (c p)
  (putprop c p 'parent)
  (putprop p
    (append (get p 'children) (list c))
    'children))

```

```

(defun intersection (x y)
  (cond ((null x) nil)
        ((member (car x) y)
         (cons (car x) (intersection (cdr x) y)))
        (t (intersection (car x) y))))

```

```

(defun addr (register value)
  (setf register (cons value (cdr register))))

```

```

(putprop 'a '(determiner singular indefinite) 'features)
(putprop 'the '(determiner definite) 'features)
(putprop 'long 'long 'adjective-function)
(putprop 'red '(adjective) 'features)
(putprop 'large '(adjective) 'features)
(putprop 'screwdriver '(noun singular) 'features)
(putprop 'screwdrivers '(noun plural) 'features)
(putprop 'long '(adjective) 'features)
(putprop 'screwdriver '(noun singular) 'features)
(putprop 'screwdriver '(s1 s2 s3 s4 s5) 'instance)
(putprop 'screwdrivers 'screwdriver 'singular-form)
(putprop 'tool '(hammer screwdriver saw wrench) 'instance)
(putprop 'hammer 'h1 'instance)
(putprop 'saw 'saw1 'instance)
(putprop 'wrench '(w1 w2) 'instance)
(putprop 'large 'large 'adjective-function)
(putprop 'red 'red 'adjective-function)
(putprop 's1 'large 'size)
(putprop 's1 'blue 'color)
(putprop 's1 '7 'length)
(putprop 's2 'large 'size)
(putprop 's2 'red 'color)
(putprop 's3 'large 'size)
(putprop 's3 'red 'color)
(putprop 's4 'small 'size)

```

```

(putprop 's5 'small 'size)
(putprop 'n1 'metal 'material)
(putprop 'saw1 'metal 'material)
(putprop 'n1 'metal 'material)
(putprop 'metal 'metal 'adjective-function)
(putprop 'metal '(adjective) 'features)
(putprop 'tool '(noun singular) 'features)
(putprop 'tools 'tool 'singular-form)
(putprop 'tools '(noun plural) 'features)
(putprop 'saw 'metal 'material)
(putprop 'hammer 'metal 'material)
(putprop 'wrench 'metal 'material)
(putprop 'screwdriver 'metal 'material)

(defun metalp (object) (equal (get object 'material) 'metal))

(putprop 'name '0 'namecounter)

(compile parse-noun-group
  (s1 (if (parse-word this-node 'determiner)
    -> s2
    after
    (setf 'number (select '(singular plural)
      (getf last-parsed)))
    (setf 'determiner (select '(definite indefinite)
      (getf last-parsed)))))
  (s2 (if (parse-word this-node 'adjective)
    -> s2
    after
    (addr 'adjectives last-parsed))
    (if (parse-word this-node 'noun)
    -> win
    after
    (setf 'number (select '(singular plural)
      (getf last-parsed)))
    (setf 'noun last-parsed))))

(compile parse-clause
  (s1 (if (parse-noun-group this-node nil)
    -> s2
    after
    (setf 'subject last-parsed)))
  (s2 (if (parse-word this-node '(verb tensed))
    -> s3
    after (setf 'verb last-parsed)))
  (s3 (if (and (equal last-parsed 't)
    (parse-word this-node 'pastparticiple))
    -> s4

```

```

after
  (setf 'object (getf 'subject))
  (setf 'subject nil)
  (setf 'verb last-parsed))
(if (and (testf (getf 'verb) 'transitive)
  (parse-noun-group this-node nil))
  -> s4
  after (setf 'object last-parsed))
(if (or (testf (getf 'verb) 'intransitive)
  (getf 'object))
  -> s4))
(s4 (if (and (setf 'subject)
  (null remaining-words))
  -> win)
  (if (and (not (getf 'subject))
    (equal current-word 'av)
    (parse-word this-node nil))
    -> s5)
  (if (not (setf 'subject))
    -> s4
    after
      (setf 'subject 'someone)))
(s5 (if (parse-noun-group this-node nil)
  -> s4
  after
    (setf 'subject last-parsed))))

```

```

(defun phillipsf (object) (equal (get object 'type) 'phillips))
(putprop 'phillips '(adjective) 'features)
(putprop 'phillips 'phillipsf 'adjective-function)
(putprop 's1 'phillips 'type)
(putprop 's4 'phillips 'type)

```

APPENDIX C

USER'S GUIDE

The English-command programs and the frame system's programs are implemented in Franzlisp for the PDP-11/70 under UNIX. UNIX terminals at the Naval Postgraduate School are located in Room 502. An account number may be acquired through the Computer Science Office from Al Wong or Bruce Montague.

Once the programs in Appendices A and B have been acquired (either by entering them yourself, or from another account), using them is straightforward. After logging on:

- (1) Enter the LISP interpreter by entering percent (%) LISP,
- (2) Load the LISP programs to the interpreter,
- (3) Execute the program "builder" to construct the frames by entering (builder), and
- (4) You are now prepared to ask questions of the system of the format presented in Chapter V of this study.

These programs are easily improved by entering new dictionary words to the program builder, and by changing commands in the program Parse-command.

LIST OF REFERENCES

1. Ackoff, R. L., "Management Misinformation Systems", Management Science, v. 14, pp. 147-56, December, 1967.
2. McKenny, J. L. and Keen, P. G. W., "How Managers' Minds Work", Harvard Business Review, v. 51, pp. 41-68, May-June 1974.
3. deBono, E., Lateral Thinking for Management, pp. 3-7, American Management Association, 1971.
4. Mintzberg, H., "Planning on the Left Side and Managing on the Right", Harvard Business Review, v. 53, pp. 49-58, July-August 1976.
5. Roland R. J., An Interactive Decision Support System for Technology Transfer Pertaining to Organization and Movement, working paper at the Naval Postgraduate School, Monterey, CA, 1980
6. Huber, George P., "The Nature of Organization Design of Decision Support Systems", v. 5, pp. 1-10, MIS Quarterly, June 1981.
7. Mintzberg H.; Duru, R.; Theoret, A., "The Structure of Unstructured Decision Processes", Administrative Science Quarterly, v. 21, pp. 246-75, June 1976.
8. Stavell, C. B., Individual Differences in Managerial Decision Making Processes: A Study of Conversational Computer System Usage, Ph.D Thesis, Massachusetts Institute of Technology, September 1974.
9. Gorry, G. A., Scott Morton, M. S., "A Framework for Management Information Systems", Sloan Management Review, v. 13, pp. 55-70, Fall 1971.
10. Simon, H. A. and Newell A., Humans as Information Processors, pp. 39-50, American Psychological Association, 1971.
11. Davis, R.; Buchanan, B. G.; Shortliffe, E. H., "Production Rules as a Representation for a Knowledge-based Consultation System", Artificial Intelligence, c. 8, pp. 14-45, August 1977.
12. Sprague, R. H. and Carlson, E. D., Building Effective Decision Support Systems, pp. 3-54, Prentice-Hall, 1982.

13. Bonczek, R. H.; Holsapple, C. W.; Whinston, A. B., "Future Directions for Developing Decision Support Systems", Decision Sciences, v. 11, pp. 616-31, January 1980.
14. Winston, P. H., Artificial Intelligence, pp. 235-52, Addison-Wesley, 1977.
15. Bonczek, R. H.; Holsapple, C. W.; and Whinston, A.B., Foundations of Decision Support Systems, Academic Press, 1981.
16. Dolk, D. R., The Use of Abstractions in Model Management, Ph.D Thesis, University of Arizona, 1982.
17. Minsky, M., "A Framework for Representing Knowledge", P. Winston Ed., in The Psychology of Computer Vision, pp. 211-77, McGraw-Hill, 1975/
18. Barr A. and Feigenbaum, E. A., The Handbook of Artificial Intelligence, v. 1, HuerisTech Press, 1981.
19. Elam, J. J.; Henderson, J. C; Miller, L. W., Model Management Systems: An Approach to Decision Support in Complex Organizations, unpublished working paper, The Wharton Schook, University of Pennsylvania, 1981.
20. Winston, P. H. and Horn, B. K. P., LISP, Addison-Wesley, 1981.
21. Goldstein, I. P. and Roberts B., "Using Frames in Scheduling", in AI: An MIT perspective, P. H. Winston and R. H. Brown eds., v. 1, MIT Press, 1979.
22. Weizenbaum, J., "ELIZA--A Computer Program for the Study of Natural Language Communication Between Man and Machine", Communications of the ACM, v. 9, January 1965.
23. Lenat, D. B., "AM: An Artificial Intelligence Approach to Discovery in Mathematics as Hueristic Search", SAIL, AIM-286, Artificial Intelligence Laboratory, Stanford University, July 1976.
24. Hillier, F. S. and Lieberman, G. F., Introduction to Operations Research, Holden-Day, Inc., 1967.
25. Tennant, H., Natural Language Processing, PBI, 1981.

INITIAL DISTRIBUTION LIST

	No. Copies
1. Defense Technical Information Center Cameron Station Alexandria, Virginia 22314	2
2. Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
3. Computer Technology Curricular Office Code 37 Naval Postgraduate School Monterey, California 93940	1
4. Lt. Col. J. K. Mullane, Code 0309 U.S. Marine Corps Representative Naval Postgraduate School Monterey, California 93940	1
5. 1LT George W. Watson, Jr. 117 Main Street Westford, Massachusetts, 01886	2
6. Professor G. Rahe, 52Ra Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
7. Assistant Professor D. D. Dolk, 54Dk Department of Administrative Sciences Naval Postgraduate School Monterey, California 93940	1

END

FILMED

9-83

DTIC